

What's New

Two new files have been added to provide better support for the Simplified Chinese language and code page 1386.: IPFGBK.NLS and APSY1386.APS.

Two files have been modified to provide better support for the Simplified Chinese language and for code page 1381: IPFPRD.NLS and APSY1381.APS.

Introducing IPF

The Information Presentation Facility (IPF) is a tool that enables you to create online information, to specify how it will appear on the screen, to connect various parts of the information, and to provide help information that can be requested by the user.

It is a tool for both the information author and the application programmer. IPF implements guidelines recommended by the Common User Access* (CUA*) element of the Systems Application Architecture* (SAA*) platform.

* Trademark of the IBM Corporation

What IPF Offers

As an author of online information, you need to know what type of information users need - tutorial, reference, or help. For example, they might need a tutorial to learn a software program, reference information for additional topics, or help information for assistance with the program.

As a programmer implementing or customizing online information, you need to know how the IPF programming interface supports your intended design.

IPF features include:

- A tagging language that formats text, provides ways to connect information units, and customizes windows
- A compiler that creates online documents and help windows
- A viewing program that displays formatted online documents
- A help manager programming interface that allows you to control and customize the display of online information.

The Tag Language

The IPF tagging language provides the instructions for how online information is to be displayed. With these instructions, or tags, you can:

- Highlight text
- Set margins
- Add lists, notes, and notices
- Create tables
- Change the size and style (font), and the color of displayed information
- Control the formatting of lines of text
- Illustrate with examples, figures, and art
- Customize windows
- Define ways to connect information units
- Establish communication links to other applications.

The IPF Compiler

When you have finished writing and tagging, information is ready to be compiled. The IPF compiler interprets the tags in your source file and

converts the information into the appropriate format. The compiler is able to distinguish between tags and text because each tag starts with a colon (:), is immediately followed by the tag name, and then ends with a period (.). For example, the tag that indicates a new paragraph is the **:p.** tag. When the compiler encounters this tag, it interprets it as, "Insert a blank line before the paragraph tag and start the text that follows the paragraph tag."

At compile time, you specify what format you want. For online documents, you direct IPF to generate a file with an INF extension. For help information, you specify a file with an HLP extension. For information about compiler commands and options, see [Starting the IPF Compiler](#).

The View Program

The View program (VIEW) enables you to display your compiled document. VIEW retrieves files with an INF extension and displays the formatted information in a standard OS/2 window.

Note: You cannot use VIEW to display files with an HLP extension. For information about how to use VIEW, see [Viewing an Online Document](#) or see the Command Reference.

The Help Manager Programming Interface

You can use IPF to develop a user interface that provides help for application windows and fields within windows. You can even customize help information by writing Presentation Manager programs that communicate with IPF.

Features of the IPF Interface

Online designs need to communicate information through a simple interface that lets the user find information quickly and easily. With IPF you can develop an interface that provides unique usability features, including:

- Hypertext links
- Push buttons
- Customized windows
- Master help index and glossary.

Hypertext and Hypergraphic Links

IPF gives the user the ability to connect to different units of text and graphics. The connections that join these units are known as hypertext or hypergraphic links. For example, a user can select a particular link to obtain related information or perhaps to see a graphical description of the topic.

An advantage of using hypertext or hypergraphic links is that the author can present information in a nonlinear way. Users can then access information both sequentially and randomly. This lets them explore or branch into subject matter that may be unclear or that needs to be reviewed. For information about hypertext and hypergraphic links, see [Linking](#).

Push Buttons

Push buttons provide users with a fast and easy way to access commonly used IPF tasks. When a user selects a push button, the action represented by the text on the push button is carried out immediately. IPF provides one set of push buttons for online documents and another set for Help windows. IPF also provides help on how to use the push buttons.

As a designer, you can change the text of a push button, select which push buttons you want to use, add your own push buttons, and specify the area of a window to place them. For more information, see [Push Buttons](#).

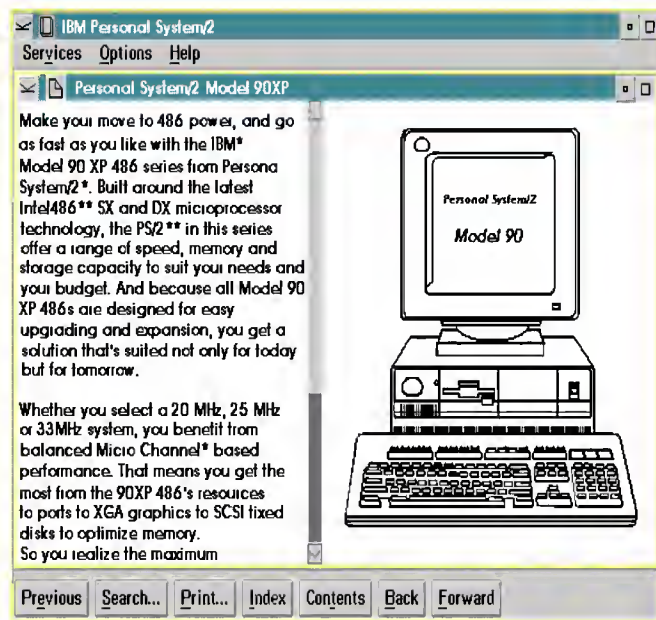
Customized Windows

A window is the area of the screen in which information is displayed. As an author of online information, you can customize windows. Different windowing effects are achieved with the IPF tagging language. For example, a window can be split so that scrollable text can be displayed beside a stationary illustration that the text describes.

The following figure shows an IPF split-window design that describes the IBM® Personal System/2® Model 90 XP 486® series.

* Trademark of the IBM Corporation

** Trademark of Intel Corporation



IPF Split Window

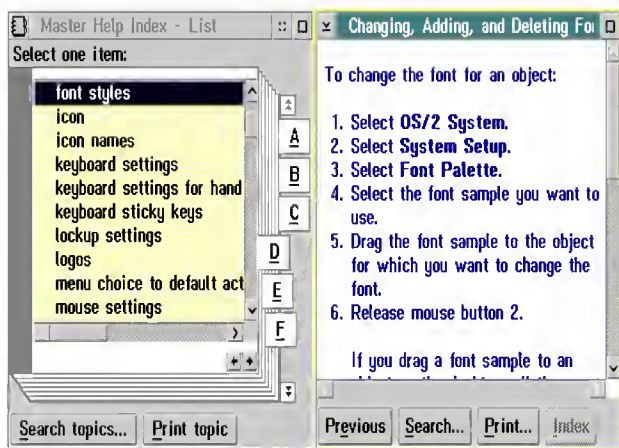
For more information, see [Customizing Windows](#).

Master Help Index and Glossary

The OS/2 operating system provides online help that can be accessed through both a Master Help Index and a Glossary. The Master Help Index and the Glossary are both collections of alphabetized pointers to information panels. Their primary purpose is to provide quick access to help topics. With them, you can provide such features as:

- Availability of the Master Help Index from anywhere in the WorkPlace Shell
- A side-by-side window design that lets the user scan index entries on one side, then display the help-text information on the other side
- A menu and buttons that let the user perform a search, print help-text windows, or request assistance.

The following figure shows an example of the Master Index window and the opened help-text window, "Changing, Adding, and Deleting Fonts."



For more information, see [Customizing Master Help Index and Glossary Objects](#)

IPF User Interface

This chapter describes the components of the IPF user interface. As an author, you will use these components when developing an online document or online help. As a programmer, you should understand the IPF user interface before you enable your application to work with online help.

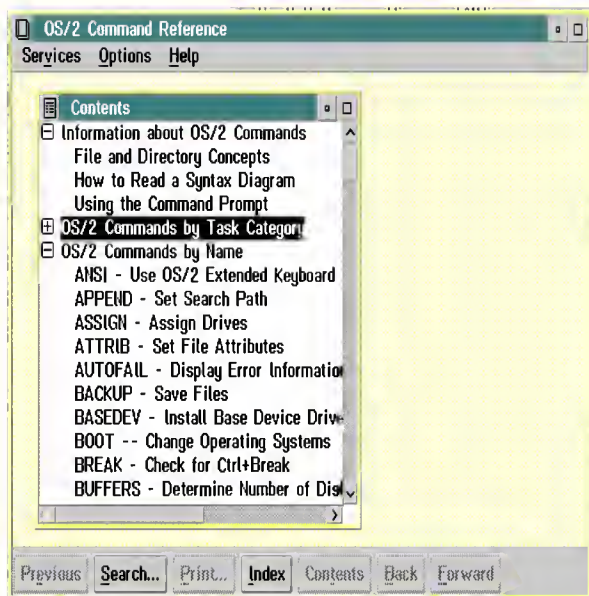
Online documents include reference or procedural information, such as converted printed material, tutorials, and organization charts. Online help includes information that users of online documents or application programs might want to access.

Because they have varying backgrounds, interests, motivations, and experiences, no two users of online information are exactly alike. To accommodate differences, IPF provides a flexible interface that can be customized according to personal preference. However, when an online document requires no special design considerations, the IPF compiler provides an automatic default design that includes:

- A Contents window
 - Standard OS/2 windows
 - Help.
-

The Contents Window

When users first select a document for viewing, IPF displays an OS/2 window that includes a table of contents (Contents window) similar to the window shown in the following figure.



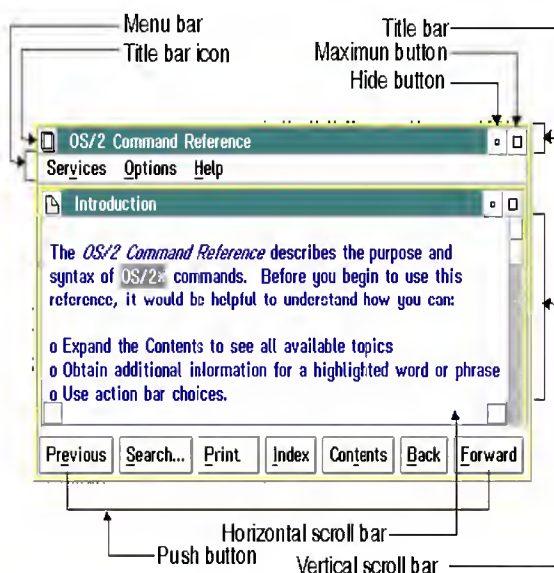
A Contents Window. Users select the highlighted item and go directly to a window of text.

The Standard Window

Unless special window characteristics are defined with IPF tags, the IPF compiler formats a window that includes the following elements:

- Menu bar
- Title bar icon
- Title bar
- Maximize button
- Hide button
- Horizontal scroll bar
- Vertical scroll bar
- Push buttons.

The following figure shows a standard window and its elements.



Standard Window and Elements

The title bar icon, and the maximize and hide buttons allow a user to change the size and position of a window. The menu bar, push buttons,

and scroll bars allow a user to work with the window's contents. The window title indicates the subject of the information, or name of the object, seen in the window.

Help

While using an online document or application program, a user occasionally requires additional information about choices, fields, or procedures for a task. CUA guidelines recommend that a product provide information to a user about how to use the product. Information about how to use a product is known as help information. The OS/2 user interface for help information is developed with IPF and is accessible from the menu bar. Help can also be accessed from push buttons located at the bottom of the window or by pressing the F1 key.

Main Help Window

When a user requests help from a window, IPF displays the main help window, the characteristics of which are:

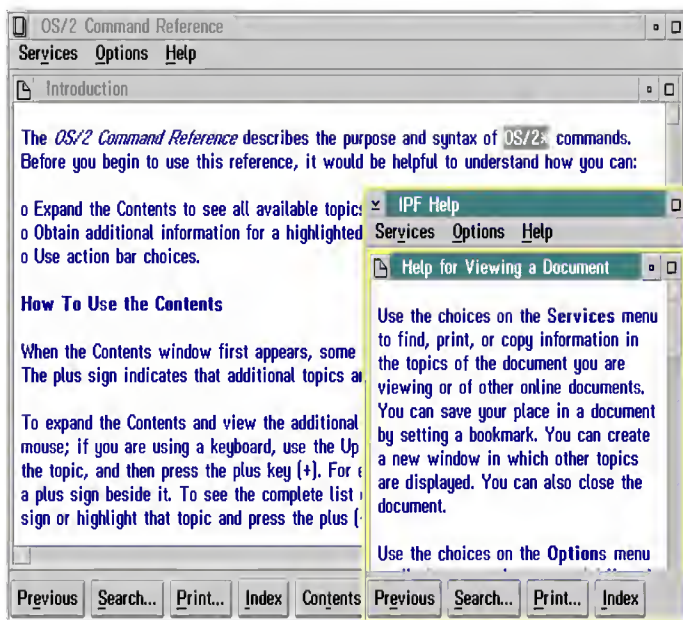
- Menu bar
- Title bar icon
- Title bar
- Maximize button
- Horizontal scroll bar
- Vertical scroll bar
- Push buttons.

The main help window cannot be minimized.

Within the main help window is the help-text window. The help-text window contains the response to the user's request for help. The characteristics of this window are:

- Title bar icon
- Title bar (shows title of the selected help window)
- Maximize button
- Hide button
- Horizontal scroll bar
- Vertical scroll bar.

The windows shown in the lower right corner of the following figure are main help and help-text windows.



An IPF Main Help Window with its Help-Text Window. The title "Help for Viewing a Document" that appears in the title bar was created by the author of the help-text window.

When the main help window is first opened, its position is such that it covers the smallest part of the online document or application window as possible. The help-text window is opened at its maximum size within the main help window.

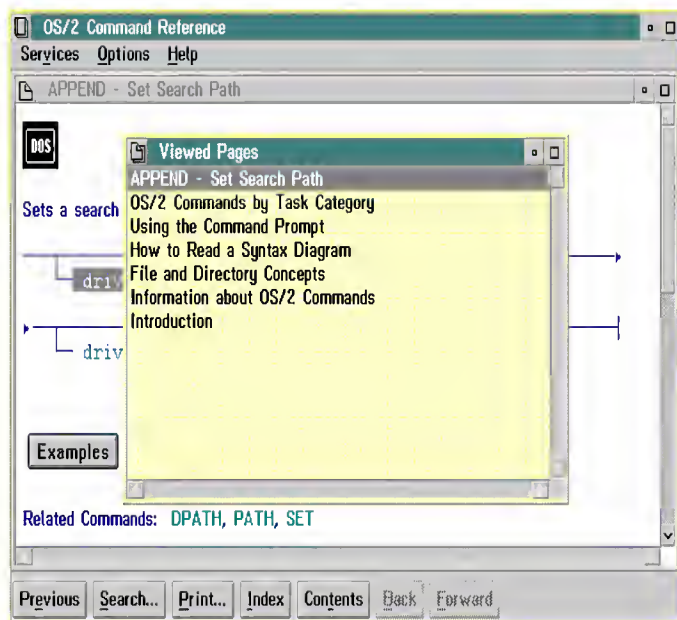
However, when the main help window is opened, it can be moved and resized, as can the help-text window. If the user makes the help-text window larger or smaller, the text within the window is reformatted to fit the new window size.

Selection Lists

Selection lists appear when some of the following menu bar choices are selected:

- **Viewed pages**, under **Options**
- **Contents**, under **Options**
- **Help index**, under **Help**
- **Libraries**

The following figure shows the Viewed Pages selection list.



View Pages Selection List

Search results also are displayed in a selection list. Selection lists differ from the help-text window in that they can be closed, either by selecting the **Close** symbol or by pressing the Esc key.

Selecting the hide button from the title bar while a help-text window, Contents window, Viewed Pages window, Index window, or Search results window is displayed, results in the window being replaced by an icon.

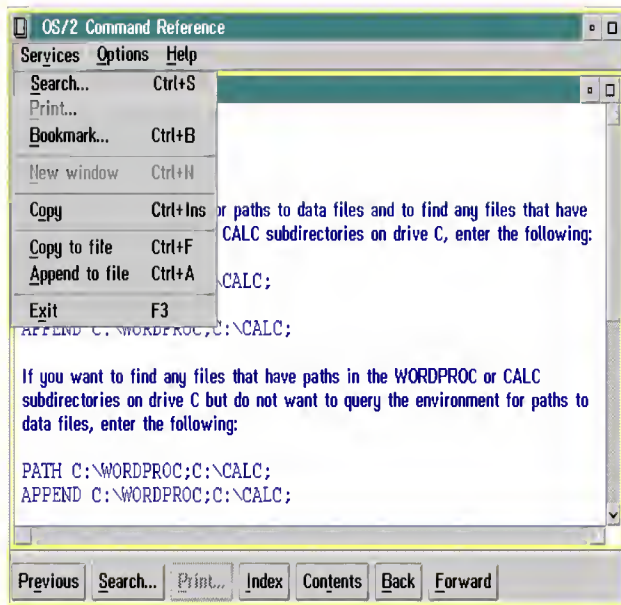
Menu Bar

The following figure shows the menu bar that has the choices **Services**, **Options**, and **Help**.

When **Services**, **Options**, or **Help** is selected, a menu appears with a list of entries that also can be selected. Some entries have an associated shortcut key or key combination. These are shown to the right of the menu item.

Services Menu

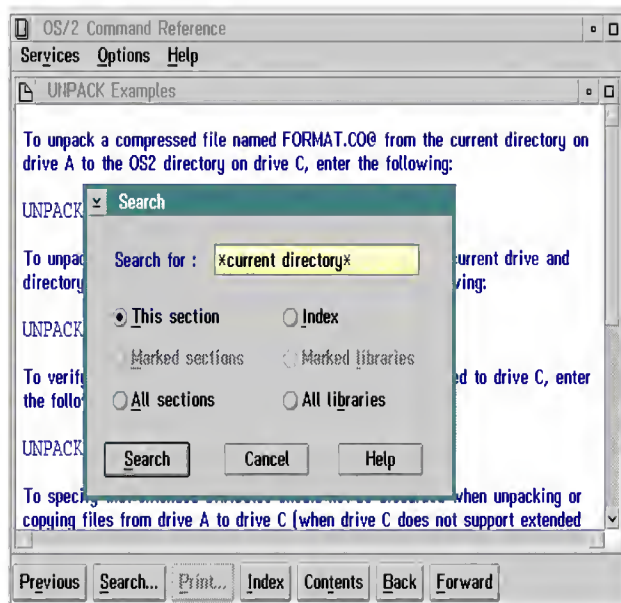
The Services menu shows a list of available services.



Services Menu

Search

This choice, and the Search push button, display the window shown in the following figure. The user can type a text string consisting of letters, numbers, blank spaces, and special characters, then select any of the choices to search for the text string.



Search Window

Global file-name characters (wildcards) can be used with the text string; for example:

current directory

The global file-name character, in this case an asterisk, automatically finds all possibilities of the *current directory* string.

Following are descriptions of Search window choices:

This section

Searches the currently displayed help-text window and highlights all occurrences of the search string that are found.

Marked sections

Searches the online-document windows or help windows whose titles were marked in the Contents window. IPF does not search unmarked secondary windows, or windows attached to the marked window by hypertext links. Before selecting **Marked sections**, the user must select **Contents** then mark the help titles to be searched. If no help titles are marked, the **Marked sections** choice is dimmed.

Sections are marked with the mouse by pressing and holding the Ctrl key then clicking mouse button 1. Sections are marked with the keyboard by using the cursor keys to highlight the item and then pressing the spacebar. The same key sequences are used to unmark the selection.

If the search is successful, IPF displays a list of the window titles where the text string was found. The search string is shown in the title bar of the search results window.

All sections

Searches the entire help library or online document and displays a list of the window titles where occurrences of the search string were found. The search string is shown in the title bar of the search results window. It does not search title text.

Index

Searches the index and displays a list of index entries in which the text string was found.

If no search string is entered, this choice displays an alphabetic list of all index topics in the help library or online document.

Marked libraries

Searches selected help libraries or online documents. The user must be in an active window of an online document or help library and follow this procedure:

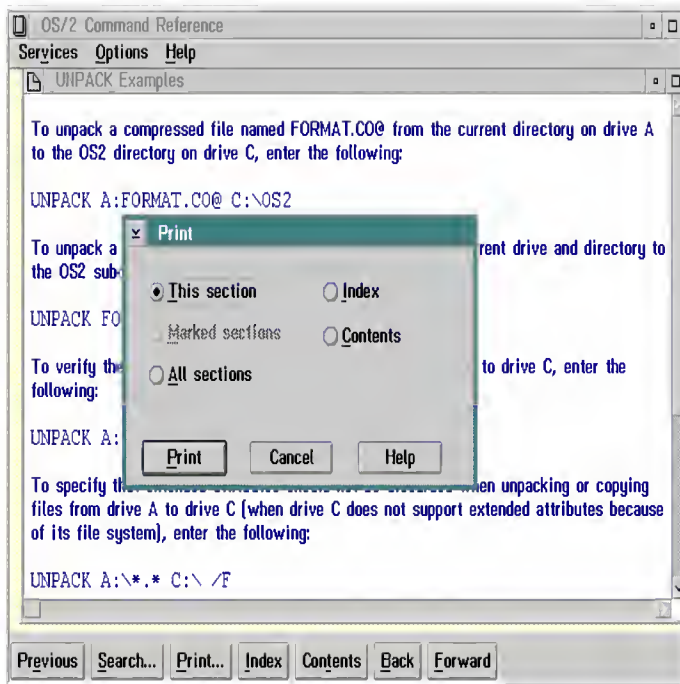
1. Select **Options** then **Libraries**.
2. Mark one or more libraries.
3. Select **Services**, then **Search**, then **Marked libraries**.
4. Select the **Search** push button.

All libraries

Searches all help libraries or online documents and displays a list of the window titles where the text string was found. The search string is shown in the title bar of the window.

Print

This choice, and the Print push button, display the window shown in the following figure. The output is the text within the window. The user can select any of the choices to print online information. All of the printed output is sent to the default printer. All of the information prints as WYSIWYG (What-You-See-Is-What-You-Get).



Print Window

Following are descriptions of Print window choices:

This section

Sends the contents of the current window to be printed.

Marked sections

Sends the sections whose titles were marked in the Contents window to be printed. Before selecting this choice, the user must select **Contents** and mark the titles to be printed. If no titles are marked, **Marked sections** is dimmed.

Sections are marked with the mouse by pressing and holding the Ctrl key then clicking mouse button 1. Sections are marked with the keyboard by using the cursor keys to highlight the item and then pressing the spacebar. The same key sequences are used to unmark the selection.

All sections

Sends the entire online document or help library to be printed.

Index

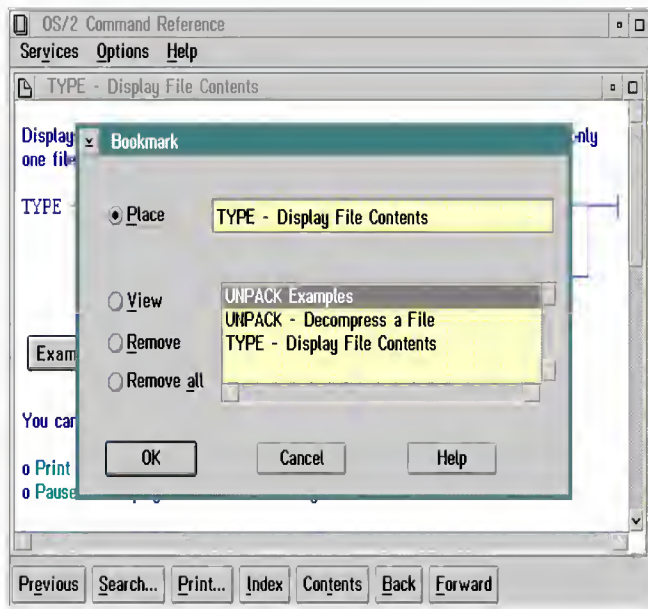
Sends the help-index or online document index to be printed.

Contents

Sends the help library or online-document contents list to be printed.

Bookmark

This choice displays the window shown in the following figure.



Bookmark Window

Place	Saves the user's place in the document being viewed.
View	Redisplays a specific place that was marked.
Remove	Deletes one marked place.
Remove all	Deletes all marked places.
Note	The Bookmark selections are saved across invocations in a file with the same name as that of the book but with a CP extension. Bookmarks are only available in online books.

New window

This choice opens a new window for the currently selected item (for example, table of content entry, hypertext link, index list, or search list) so the user can see more than one topic displayed at the same time. The user can resize the new window so the previous window also can be seen, then resize both to view both windows.

Copy

Copies the window currently being viewed to the system clipboard. The user can then select **Paste** from the menu of the OS/2 System Editor (or any other editor with this capability) to view or edit the information.

Copy to file

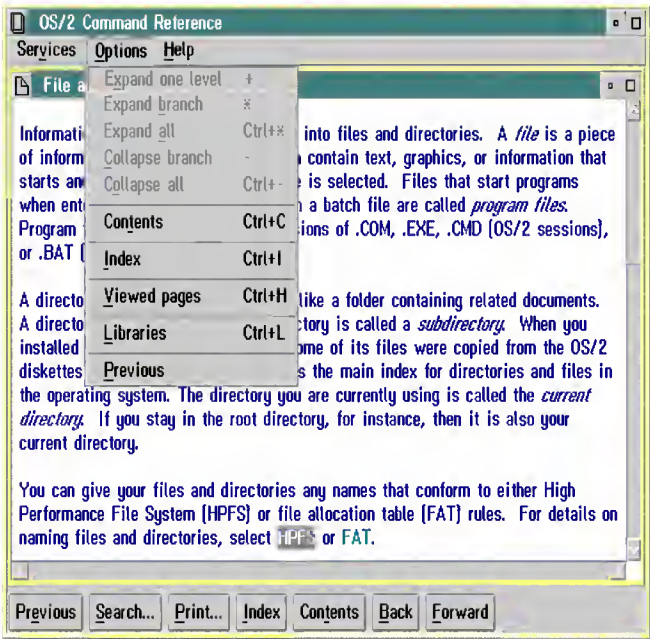
Copies the information to the file, TEXT.TMP. This file is placed in the current directory. If TEXT.TMP already exists, it is replaced.

Append to file

Copies the information to the file, TEXT.TMP. This file is placed in the current directory. If TEXT.TMP already exists, the new information is added to the existing information.

Options Menu

Selecting **Options** displays the menu shown in the following figure.



Options Menu

The first five choices are active when the Contents window is active. If the Contents window is not active, these choices are dimmed. These choices control how the table of contents will be displayed.

A tree-structured table of contents is created if more than one heading level is specified with IPF heading tags when the windows are created. (For a description of heading tags, see [Headings](#).)

If there are additional entries under a heading, "+" appears to the left of the entry. When an entry is expanded one level, the next level of entries subordinate to the selected entry is displayed, and the "+" is replaced by a "-". The user can click on the "+" or "-" symbols to expand or contract the contents.

Expand one level

Expands the first level subordinate to the selected entry.

Expand branch

Expands all levels subordinate to the selected entry.

Expand all

Displays the entire tree structure of the contents.

Collapse branch

Contracts all levels subordinate to the selected entry.

Collapse all

Displays only the highest level entries in the contents.

Contents

Lists the table of contents for the document you are viewing. You can select from the topics shown. This choice has the same function as the **Contents** push button.

Index

Displays an alphabetic list of the topics in the document. You can select from the index entries shown. This choice has the same function as the **Index** push button.

Viewed pages

This selection displays a list of all windows viewed during the current session. Window titles are listed in the order that windows were viewed.

If a window is viewed more than once, its title appears as many times as it was viewed.

The maximum number of entries that can appear in the Viewed Pages window is 50.

Libraries

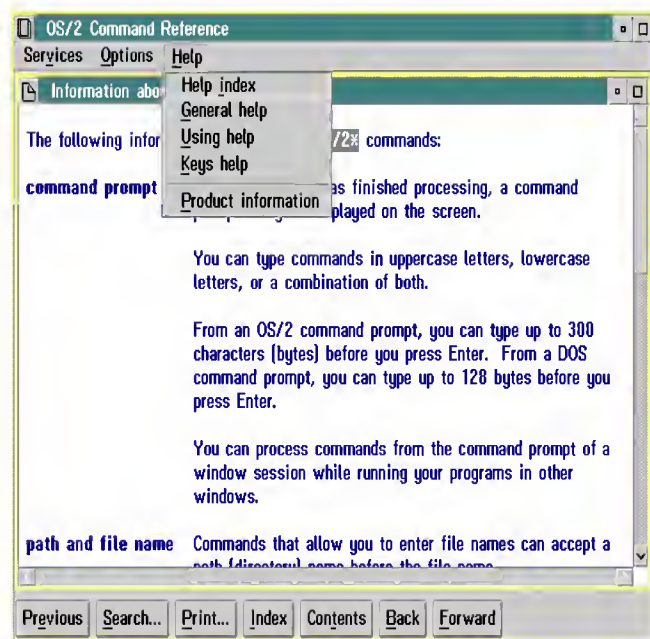
Displays a list of online libraries that are available. The libraries are identified by their directory path. The help files are the directories identified in the HELP environment variable. The online books are the directories identified in the BOOKSHELF.

Previous

Displays the previously viewed window. Each time **Previous** is selected, the previously viewed window is displayed, until the first window viewed in the current session is shown. Selecting **Previous** again from a help window, ends the current session. This choice has the same function as when the user selects the **Previous** push button; or presses the Esc key.

Help Menu

Selecting **Help** displays the menu shown in the following figure.



Help Menu

Help index

Displays an alphabetic list of topics for which help information is provided.

General help

Displays general information describing the user interface and how to access it.

Using help

Displays a list that describes the different help information that is available to users of the OS/2 operating system.

Keys help

Displays information describing key assignments for the application.

Note: Providing Keys help is the responsibility of the application programmer. A simple help window can be created that lists each key combination assigned to an application function, and a brief description of what the function does.

Product information

Displays copyright information. Providing product information also is the responsibility of the application programmer.

Tutorial

This choice is included in the Help menu if your application tells IPF that it has created a Presentation Manager tutorial application. This choice has the same function as the **Tutorial** push button.

When the user selects **Tutorial**, IPF sends a message to the application that the selection has been made. The application then starts the tutorial.

Push Buttons

Push buttons provide users with a fast and easy way to access commonly used IPF functions. When a user selects a push button, the action

represented is carried out immediately. IPF provides the following set of push buttons:

Previous

This push button lets the user see information from the previously viewed window. This is the same function as when the user selects **Options** then **Previous** from the menu bar, or presses the Esc key.

Search

This push button displays a window that lets the user search for a word or phrase. This is the same function as when the user selects **Services** then **Search** from the menu bar.

Print

This push button displays a window that lets the user print one or more topics. This is the same function as when the user selects **Services** then **Print** from the menu bar.

Index

This push button displays an alphabetic list of the index topics in a help library or an online document. This is the same function as when the user selects **Options** then **Index** from the menu bar.

Contents

This push button displays the Contents window. This is the same function as when the user selects **Options** then **Contents** from the menu bar.

Back

This push button displays the previous page in the table of contents hierarchy.

Forward

This push button displays the next page in the table of contents hierarchy.

Tutorial

This push button is included only if a tutorial was specified in your application. This is the same function as when the user selects **Help** then **Tutorial** from the menu bar.

Starting with the Tag Language

As the author of online information, you can use the IPF tag language to define various characteristics of text format. You also can use tags to define characteristics of the window in which the text is displayed.

There are 45 tags (excluding symbols) that make up the IPF tag language. The tags are *mnemonic*, making it easy to associate them with their functions. However, before you can begin to use this language, you need to familiarize yourself with the elements that make up the syntax of the tags, and special rules that govern the use of the tags.

Syntax Conventions

Each tag must start with a colon (:) and end with a period (.). (The period is also known as a *delimiter*.) For example, the tag for a paragraph is:

`:p.`

A tag indicates how the text that immediately follows it is to be processed. In the following example, the text immediately after the paragraph tag (`:p.`) is the actual text that is displayed in the window, and it will begin a new paragraph.

`:p. There are fewer than 1200 manatees...`

End Tags

Some tags require end tags. An end tag is **e** immediately followed by the tag. For example, the end tag for the `:userdoc.` tag is:

`:euserdoc.`

Most of the tags that have end tags affect text format or appearance. The end tag tells the IPF compiler to end the operation associated with the tag. If you forget an end tag, the compiler displays an error message.

Nested Tags

Nested tags are tags within other tags. For example, a common way of presenting information is in a list form; a tag begins the list, another tag identifies each list item, and yet another tag ends the list. An example of the tagging for a simple list follows:

```
:s1.  
:li.List item 1  
:li.List item 2  
:li.List item 3  
:es1.
```

The list-item tag (:li.) is required for each item in the list. The :li. tags are *nested* between the :s1. tag and the :es1. tag.

Note: After paragraph and heading tags, you will probably use list tags most often. IPF provides general-purpose lists (simple, unordered, and ordered), and special-purpose lists (definition and parameter).

Text Strings

Some tags have text strings associated with them. The string can immediately follow the tag, or it can start the line immediately following the tag. For example, the tagging for the title bar of a window is :h1. (one of the heading tags) and a text string, which is called a *title string*. You can enter it like this:

```
:h1.Save the Manatee
```

or like this:

```
:h1.  
Save the Manatee
```

Attributes

A tag also can have one or more *attributes*. An attribute contains additional information about a tag's operation. The attribute has a name, which may have a value or keyword assigned to it.

In the following example, the attribute **res=** specifies a window identifier.

```
:h1 res=001.Save the Manatee
```

In this case, **001** is the assigned value. The value assigned to a **res=** attribute must be unique for each heading tag. This value also will be the identifier for linking to the heading from elsewhere in the information. The concept of linking is described in [Hypertext Links](#).

Notice that the period follows the attribute, not the heading tag. The period always follows the last attribute in the tag.

You can specify many attributes in one tag, and they can extend over several lines. However, you cannot split an attribute. For example, you cannot put the **res=** attribute of the heading tag on one line, and its value, **001**, on the next line.

Some attributes are optional and have a default (an assumed value) if they are not included with the tag; other attributes are required. Tag attributes can be specified in any order.

As mentioned, some attributes are required. For example, if you are creating a help library, the **res=** attribute of a heading tag is required as a window identifier (see [Window Identifiers](#)).

An attribute also can have a keyword associated with it. For example, an attribute of the **:color.** tag is **fc=** (foreground color), which is used to specify the color of the text. Its value can be equal to any of the following keywords:

- DEFAULT
- BLUE
- CYAN
- GREEN

- NEUTRAL
- RED
- YELLOW.

Not all attributes have values or keywords. For example, if you want a simple list with no blank lines between the list items, add the **compact** attribute to the simple-list tag (:sl.). In the following example, notice the **compact** attribute stands by itself:

```
:sl compact.
:li.List item 1
:li.List item 2
:li.List item 3
:esl.
```

Attribute Values with Blank Spaces

If an attribute value includes blank spaces, the value must be enclosed in single quotes. For example:

```
:font facename='Tms Rmn'.
```

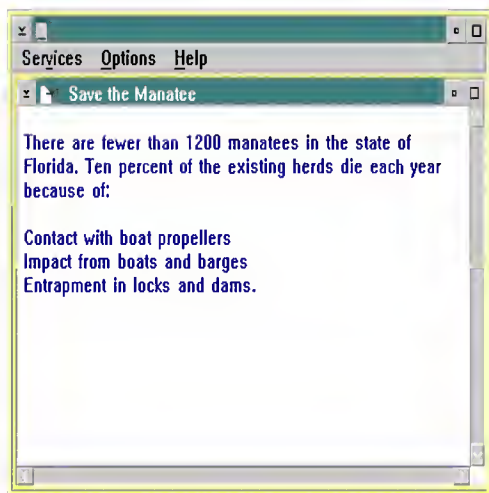
Notice that the value has initial capitals. For this particular case, they are required; otherwise, the IPF compiler will not recognize them as valid values.

Using some of the tags described thus far, you could produce a source file like this:

```
:userdoc.
:h1 res=001.Save the Manatee
:p.
There are fewer than 1200 manatees in the state of Florida.
Ten percent of the existing herds die each year
because of:
:sl compact.
:li.Contact with boat propellers
:li.Impact from boats and barges
:li.Entrapment in locks and dams.
:esl.
:euserdoc.
```

The output produced from the source file is an OS/2 standard window.

The menu-bar choices, **S**ervices, **O**ptions, and **H**elp are provided automatically by IPF. The title-bar line, "Save the Manatee," is generated by the :h1. tag. The viewing area of the window displays the formatted information.



An OS/2 Standard Window Produced from Source File

The best way to learn about tags is to study the examples provided in the following sections, then create some windows of your own.

Symbols

You use symbols to produce characters that cannot be entered from the keyboard. A symbol begins with an ampersand (&) and is followed by the symbol name and a period. For example, to produce a square bullet, which looks like this:

Enter the symbol like this:

`&sqbul.`

If you want the ampersand character (&) to appear in text, define it as the symbol, **&**. Otherwise, the IPF compiler tries to interpret whatever text follows the ampersand character as the name of a symbol, and will return the error message, `Invalid symbol.`

Symbols are case-sensitive. That is, if you do not type them exactly as they appear in the symbols table (see [Symbols](#)) you could get either the message, `Invalid Symbol`, or a symbol different from the one you want.

Note: The symbols table is also available online when you install the Online Information component of the Developer's Toolkit for OS/2.

Headings

Perhaps the most versatile tag is the heading tag. Heading tags enable information to be displayed in windows, control entries in the Contents window, control placement of push buttons in a window, and define the shape and size of windows. With IPF, you can specify six levels of headings, `:h1` through `:h6`. For information about default heading levels that start a window and place entries in the table of contents window, see the following topic on Controlling Entries in the Contents Window.

Displaying Window Titles

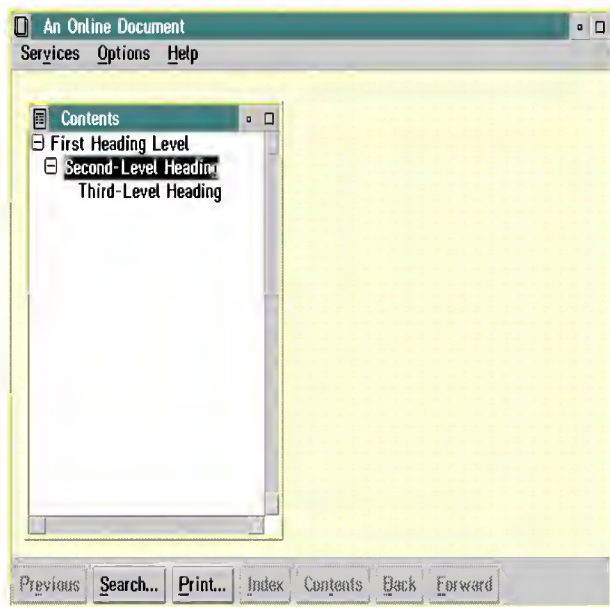
Every heading tag that starts a window must have an associated text string. The text string becomes the window title and appears in the title bar of the window. The window title also becomes an entry in the Contents window, which lists the headings of all topics in an online document.

For a window that occupies the full width of the screen, the maximum length of a text string, including spaces and blanks, is 70 characters. A narrower window requires a shorter text string. The text string can be on the same line as the heading tag, or at the beginning of the next line.

The following example shows the tagging for the first three heading levels, with a paragraph following each heading.

```
:userdoc.  
:title.An Online Document  
:h1.First Heading Level  
:p.  
This window is defined by a first-level heading tag.  
:h2.Second-Level Heading  
:p.  
This window is defined by a second-level heading tag.  
:h3.Third-Level Heading  
:p.  
This window is defined by a third-level heading tag.  
:euserdoc.
```

The Contents window for the formatted output shows the three heading-level entries.



A Contents Window

Hiding Window Titles

If you do not want a title to appear in the Contents window, use the **hide** attribute. The heading definition would be entered like this:

```
:h3 hide.
Another Third-Level Heading
```

Note: Your source file must contain at least one heading tag without the **hide** attribute.

Controlling Entries in the Contents Window

The following example shows some tagging that will control what entries appear in the Contents window, as well as what headings will start windows.

```
:userdoc.
:docprof toc=12.
:h1.Heading Levels
:h2.Second-Level Heading
:p.
This window is defined by a heading-level 2 tag.
:h2.Second-Level Heading
:p.
This window also is defined by a heading-level 2 tag.
:p.
:h3.Third-Level Heading
:p.
Because the &colon;docprof. tag at the beginning of the file
specifies that only heading levels 1 and 2 can be entries in the
Contents window (toc=12), the preceding "Third-Level Heading"
and THIS text, which follows it, become part of the
window defined by the preceding heading-level 2 tag.
:h2 toc=123.Another Second-Level Heading
```

```

:p.
The heading-level 2 tag for this window contains
a toc=123 specification.
:h3.Third-Level Heading
:p.
Because the toc=123 in the preceding heading-level 2
tag overrides the toc=12 in the &colon.docprof. tag, this
heading-level 3 tag defines a new window and creates a
Contents entry.
:euserdoc.

```

Unless otherwise specified, the default set of heading tags that create entries in the Contents window and define the start of windows are **:h1.**, **:h2.**, and **:h3.**. To change this default, specify a numeric sequence with the table of content attribute (**toc=**) of the **:docprof.** tag. The **:docprof.** tag controls the heading levels displayed in the Contents window. The sequence must begin with level 1 and cannot skip a level in the descending hierarchy. For example, the **:h4.**, **:h5.** and **:h6.** tags do not start separate windows, but control the appearance of the text of the window unless you specify:

```

:docprof toc=123456.

```

To specify that only heading levels 1 and 2 are to define windows and appear as entries in the Contents window, the following tag was used:

```

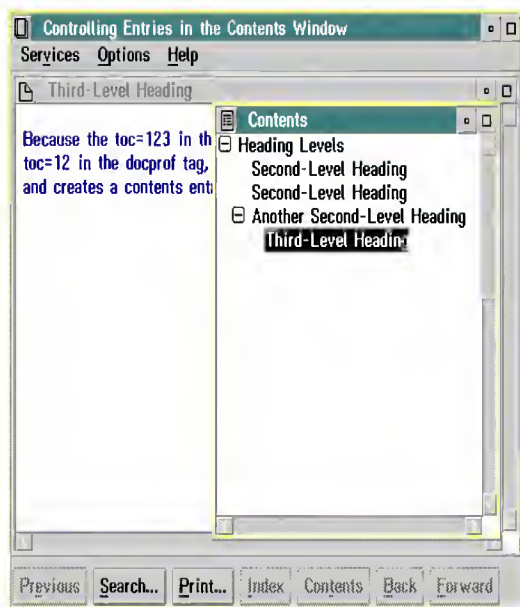
:docprof toc=12.

```

The value specified for the **toc=** attribute remains in effect for all the heading definitions in the file. You can override it by specifying another value for the **toc=** attribute in a heading definition. The new value is then in effect for the rest of the headings in the file, or until overridden in another heading definition.

In the preceding example, the **toc=** attribute of the **:docprof.** tag is overridden by the **toc=** attribute of a heading tag.

The next example shows the results of the tagging. Notice the effect of including a heading level that is lower in the hierarchy than the range of heading levels specified with the **:docprof.** tag.



Contents Window with Displayed Third-Level Heading Window

When the file is viewed, the **:h3.** title and the text following it are included as part of the window defined by the preceding **:h2.** tag.

Special Rules

Sequential Coding for Heading Tags: Headings for a series of windows must always start with **:h1.** and proceed in sequence. That is, you cannot have **:h1.** followed by **:h3.**. However, you can follow **:h3.** with **:h1.**.

Source File Size between Heading Tags: Do not exceed 16 000 words, numbers, and punctuation marks between two consecutive heading tags in your source file. This includes blank spaces, but does not include commented lines (see [Comment](#)). If the source file exceeds this limit, the compiler will generate an error message. To correct the error, use another heading tag.

For more information about heading tags and attributes that define characteristics of windows, see [Customizing Windows](#).

Note: At least one `:h1.` must not be hidden.

Push Buttons

Push buttons provide users with a fast and easy way to access commonly used IPF tasks. When a user selects a push button, the action represented by the text on the push button is carried out immediately. Push buttons are displayed in a window called a *control area*. A control area can be defined within the IPF coverpage window, or the IPF text window (the child of the coverpage window), or both. For information about the IPF coverpage, see the following figure.

IPF provides one set of push buttons for online documents and another set for help windows.

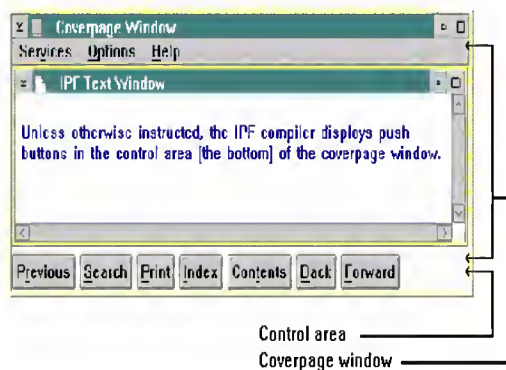
For online documents, the set of push buttons consists of:

Previous
Search
Print
Index
Contents
Back
Forward
Tutorial (only if a tutorial is available).

For help windows, the set of push buttons consists of

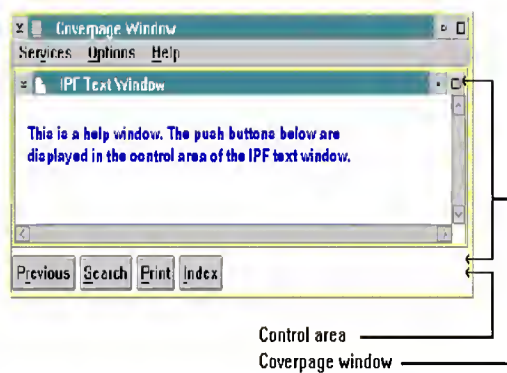
Previous
Search
Print
Index
Tutorial (only if a tutorial is available).

The following figure shows an online document with a set of push buttons in the control area of the the coverpage window (the default control area).



The IPF Default Window for Push Buttons. These push buttons appear in the control area of the coverpage window.

Notice the difference in the following figure. This example shows a help window with a set of push buttons in the control area of the IPF text window.



A Help Window. These push buttons were defined in the control area of the IPF page window.

If the user changes the size of the window, the push buttons in the control area will wrap around onto the next line. The push buttons cannot be clipped or scrolled horizontally, because the control area is not part of the scrollable area of the IPF text window.

Tagging Example for the Default Set of Push Buttons

The following example shows the minimum tagging required for an online document that is to have a control area with the default set of push buttons displayed in the coverpage window.

```
:userdoc.
:title.Coverpage Window
:h1.IPF Text Window
:p.Text goes here.
:euserdoc.
```

Notice no extra tagging is necessary.

Specifying Push Buttons for the Control Area of a Window

The control area tag (:ctrl.) specifies where push buttons are to be displayed, and which push buttons you want displayed. When specifying a control area, always precede the tagging with :docprof., then imbed :ctrl. between the control-area definition tags (:ctrldef. and :ectrldef.). For example:

```
:docprof toc=123.
:ctrldef.
:ctrl.
:ectrldef.
```

Attribute Values for the Control Area of a Window

The **controls=** attribute of :ctrl. identifies the push buttons that you want in the control area of a window. Push buttons are displayed in the order in which they are defined. Values that can be specified are:

Search	Specifies the Search push button. When selected, this push button displays a window that lets the user search for a word or phrase.
Print	Specifies the Print push button. When selected, this push button displays a window that lets the user print one or more topics.

Index	Specifies the Index push button. When selected, this push button displays an alphabetic list of the topics in the document.
Contents	Specifies the Contents push button. When selected, this push button displays the Contents window.
Esc	Specifies the Previous push button. When selected, this push button lets the user see information from an earlier request.
Back	Specifies the Back push button. When selected, this push button displays the previous page in the table of contents hierarchy.
Forward	Specifies the Forward push button. When selected, this push button displays the next page in the table of contents hierarchy.

Note: A value for the **Tutorial** push button is not provided because it is displayed automatically if a tutorial exists.

Both the **page** and **coverpage** attributes of **:ctrl** affect where push buttons are displayed. For example, you use **page** to specify that push buttons are to be in the IPF text window; similarly, you use **coverpage** to specify that push buttons are to be in the IPF coverpage window.

A control area also can have a value associated with it. The **ctrlid=** attribute specifies the value, which can be either alpha or alphanumeric, and is referred to by a heading tag. In the following example, **ctrlid=** specifies a window identifier, and instructs the compiler to display the **Previous**, **Forward**, and **Back** push buttons in the control area of the coverpage window:

```
:docprof toc=123.
:ctrldef.
:ctrl ctrlid=new1 controls='ESC FORWARD BACK' coverpage.
:ectrldef.
```

Conversely, the following example shows the tagging for an online document that will display the **Previous**, **Forward**, and **Back** push buttons in the control area of an IPF page window.

```
:docprof toc=123 ctrlarea=page.
:ctrldef.
:ctrl ctrlid=new1 controls='ESC FORWARD BACK' page.
:ectrldef.
```

Notice the **ctrlarea=page** attribute of **:docprof**.. When the IPF compiler encounters **ctrlarea=page**, it defines the control area as the IPF page window and removes the push buttons from the control area of the coverpage window. You must ALWAYS specify the **ctrlarea=** attribute in **:docprof** when overriding the default control area in a window.

Other values for **ctrlarea=** are:

coverpage	Identifies the control area as the bottom of the coverpage window. This is the default value.
both	Specifies both the control area within an IPF text window, and the coverpage window.
none	Specifies that you do not want a control area (that is, you do not want push buttons).

You can define more than one control area with different sets of push buttons for the IPF text window; however, only one set of push buttons can be defined for the coverpage window.

Controlling the Display of Push Buttons in Designated Windows

Suppose your document consisted of 100 windows, and you wanted only one window to display push buttons in the control area of the IPF text window. The **ctrlarea=** attribute of a heading tag specifies which control area in a window you want to display push buttons. You would tag your source file as follows:

```
:docprof ctrlarea=none.
.
.
.
:h1 ctrlarea=page.One Window
```

When **ctrlarea=** is encountered in a heading tag, it overrides the **ctrlarea=** attribute specified by **:docprof**..

Disabling the Display of Push Buttons

The following example shows the minimum tagging for an online document without push buttons.

```
:userdoc.  
:title.Coverpage Window Title  
:docprof toc=123 ctrlarea=none.  
:hl.IPF Text Window  
:p.Text goes here.  
:euserdoc.
```

Author-Defined Push Buttons

IPF also supports author-defined pushbuttons. For example, you can define a push button for **Examples** that can be included in the control area of a coverpage or IPF text window. When an author-defined push button is selected, the message HM_NOTIFY is sent to the application or communication object. It is the responsibility of the application or communication object to respond to this message. For information about communication objects, see [Controlling Windows with Applications \(ACVIEWPORTS\)](#).

The push button tag (:pbutton.) defines author-defined push buttons. This tag must be imbedded within the :ctrldef. and :ectrldef. tags, and it must precede the :ctrl. tag.

The following example shows how to override the default set of push buttons in the coverpage window with a set that consists of **Search**, **Index**, **Previous**, and **Example**.

```
:userdoc.  
:docprof toc=123 dll='example' objectname='xmpbutton'.  
:ctrldef.  
:pbutton id=xmp res=257 text='~Example'.  
:ctrl ctrlid=new1 controls='SEARCH INDEX ESC XMP' coverpage.  
:ectrldef.
```

Notice that a dynamic link library (DLL) is required to support the function you want to provide with an author-defined push button. For more information, see [:pbutton. \(Push Button\)](#).

About the Tutorial Push Button

When the **Tutorial** push button is selected, the message HM_TUTORIAL is sent to the application or communication object. This is the same message that is sent when the **Tutorial** choice is selected from the Help pull-down, or when the **tutorial** attribute is specified with the heading tag.

The **Tutorial** push button is included only if a tutorial was specified in the initialization structure (HMINIT) or with the tutorial attribute in a heading tag.

Indexing

IPF provides an index for both online documents and help windows from the following tags.

```
:i1.  
:i2.
```

The :i1. tag creates a *primary entry*, which means the entry is at the first level. The :i2. tag provides a secondary entry to the primary one.

Index entries are imbedded in the text of a window. You should create at least one index entry for each window, using the :i1. tag. The text of

an index entry must be on the same line as the tag.

You form an index for online documents and help windows the same way. For example, to create the index entry:

```
copy program
```

use the following tagging

```
:i1.copy program
```

To create two levels of index entries, you use the **:i1.** tag with the **id=** attribute, and the **:i2.** tag, with the **refid=** attribute. Here is how to do it.

1. Create the primary index entry and give it an identifier; for example:

```
:i1 id=prnt.printers and plotters
```

2. Create the secondary index entries that will be listed under the primary index entry, and refer to the identifier of the primary entry; for example:

```
:i2 refid=prnt.change printer  
:i2 refid=prnt.add printer  
:i2 refid=prnt.printer properties
```

When an **:i1.** tag has an identifier that is referred to by **refid=** attributes of **:i2.** tags, the **:i1.** tag must precede the **:i2.** tags in the file. Index entries can be located in any of the windows defined in your source file; however, they cannot be in a footnote.

After your source file is compiled and the user selects **Index** from the **Options** menu, or the **Index** push button, the index entries look like this:

```
printers and plotters  
  add printer  
  change printer  
  printer properties
```

Index-Synonyms

As a way of helping the user search for index entries by using synonyms, IPF provides the index-synonym tag (**:isyn.**). This tag requires the **root=** attribute. With these, you can specify synonyms that will be associated with primary index entries. The **:i1.** tags for these primary entries require a **roots=''** attribute that associates the entry with the synonyms.

For example, assume you have the following entries in your file:

```
:isyn root=copy.  
copy copying duplicate duplicating  
:isyn root=folder.  
folder folders document documents  
:i1 roots='copy folder'.  
copying a document
```

The **roots=''** attribute of the **:i1.** tag associates "copying a document" with the synonyms of the **root=** attributes of the two **:isyn.** tags.

Now if a user, when requesting a search of the index, specifies any of the words in either of the two **:isyn.** entries, the search results will include all **:i1.** entries that contain the specified word, as well as any **:i1.** entries that have been associated with the word by a **roots=** attribute.

For example, the user enters "duplicating" in a search request. When the search is completed, one of the entries in the search results window is:

```
copying a document
```

Customizing Master Help Index and Glossary Objects

OS/2 online information is accessible through the "Master Help Index" and "Glossary" Workplace Shell objects. Users can access alphabetized lists of index entries simply by double clicking on the icons for these objects.

As an author who wants to create or modify Help Indexes and Glossaries, you need to understand that the Master Help Index object and Glossary object that are installed by default with the OS/2 operating system are the same class of object. In other words, they are the exact same *kind* of object. Like the rest of the Workplace Shell user objects, they can be created, shadowed, copied, and so on. This also means that they can be modified through the "Settings Notebook". To look at the Settings Notebook for the Master Help Index, simply do the following:

1. Select the Master Help Index user object by single-clicking on its icon with mouse button 1.
2. Press mouse button 2 for the object's pop-up menu.
3. Select "Open" with mouse button 1, then select "Settings" from the pop-up menu that appears.

The "Properties" page of the Master Help Index object's Settings Notebook can be modified. On this page, the "Files/Environment name(s):" setting can be modified. This setting specifies where online helps (.HLP files) containing Master Help Index entries are located. The Master Help Index object's default setting is defined in the file **CONFIG.SYS** by the "HELP" environment variable.

The **global** attribute of the **:i1.** and **:i2.** tags identifies index entries as candidates for the Master Help Index or Glossary. Good candidates are pointers to procedural and conceptual topics. For example, a simple Master Help Index entry for conceptual information about batch files might look like this:

```
:i1 global.batch files, creating
```

When referring to an **:i1.** tag, use the **global** attribute in both the **:i1.** and **:i2.** tags. For example:

```
:i1 id=copy global.copying
:i2 refid=copy global.help topics
:i2 refid=copy global.document topics
```

As an author, you can create product-specific indexes and glossaries, add entries to the Master Help Index or Glossary (or customized, renamed copies of either user object), change their Notebook "Files/Environment name(s):" Settings to match online helps with **global** index entries, and so on. However, it requires programming skills to enable applications to do these tasks at installation or run time (see [Creating Master Indexes and Glossaries with Applications](#)).

Control Words

In addition to tags, you can include *control words* in your source files to request special processing from the IPF compiler. A control word is placed at the beginning of a line, and starts with a period (.). Control words must begin in column one.

The IPF compiler recognizes the following control words:

.im <i>filename</i>	Imbed this file in the current file.
.*	Treat this line of text as a comment and do not interpret.
.br	Start a new line of text.
.nameit	Perform compile time symbol substitution.
.ce	Centers Text

Imbed

The IPF compiler can produce a single output document by processing one master source file that *imbeds* other source files. The imbed control word (**.im**) sends a signal to the compiler to process each file in the sequence listed in the master file.

This process is most often associated with online documents. A portion of the master file for the online *IPF Reference* looks like this:

```
:userdoc.  
.  
.  
.  
.im ipfcch01.ipf  
.im ipfcch02.ipf  
.im ipfcch03.ipf  
.  
.  
.
```

If you are imbedding files, the source file that begins with the **:userdoc.** tag is considered the master file. The imbedded files cannot have **:userdoc.** and **:euserdoc..**

Comment

Occasionally, you might want to insert comments in your source file solely for the purpose of providing information. The **.*** enables you to do this. Any text on the same line as this control word is ignored by the compiler. For example, the compiler would recognize the following lines as comment lines and ignore them.

```
.*  
.* *****  
.* This file contains the  
.* introduction to IPF.  
.* *****
```

Break

The break control word (**.br**) interrupts the display of text on a line, and continues it on the next line. The break control word must be the only entry on the line. For example, assume the source file has the following lines.

```
:p.These words  
appear on  
the same line.  
.br  
These words  
.br  
do not.
```

The output looks like this:

```
These words appear on the same line.  
These words  
do not.
```

If you enter text on the same line as the break control word, the IPF compiler ignores the break control word.

Nameit

The nameit control word (**.nameit**) is a macro that allows you to create a special kind of symbol. Nameit has two operands that are used in IPF, SYMBOL and TEXT. The operands can be in any order (**.nameit symbol=name text='string'**).

Symbol=name identifies the name of the symbols (just the name, without the preceding & or the trailing period) you want to create. It can be

up to 10 characters long (A-Z, 0-9), with no blanks or special characters; the first character must be a letter.

Text='string' identifies the content of the value to be assigned to the symbol and is what is printed.

Here's an example of how **.nameit** could be used in your document:

```
.nameit symbol=os text='operating system'
.
.
:p.The &os supports multitasking.
```

The output looks like this:

```
The operating system supports multitasking.
```

Note: You do not use the & in the **.nameit** when you actually use the symbol. Also, you must use a period at the end of the symbol name when you use the symbol.

Center Text

The **ce** control word (**.ce**) enables you to center text. For example, assume the source file has the following lines.

```
:p.These words
appear as normal
text on the same
line.
.ce These words appear as centered text on the same line.
```

The output looks like this:

```
These words appear as normal text on the same line.
                These words appear as centered text on the same line.
```

Displaying Text and Graphics

Once you have defined your window, you need to consider the various ways text can be displayed. This chapter describes how you can use tags and symbols to:

- Highlight text
- Add notes, notices, and lists
- Define tables for a structured display of data
- Illustrate your text with examples, figures, and character graphics
- Control the formatting of lines of text
- Change the font and color of the displayed information
- Set the margins of the text
- Display art.

Highlighted Phrases

Text can be highlighted by using different type styles or color. There are nine highlighted-phrase tags you can use to emphasize text (:hp1. through :hp9.). Each tag requires a corresponding end tag (:ehp1. through :ehp9.).

In the following example, the highlighted phrases are shown as list items in a compact simple list.

Input Example

```
:sl compact.
:li.:hp1.Highlighted phrase 1 looks like this.:ehp1.
:li.:hp2.Highlighted phrase 2 looks like this.:ehp2.
:li.:hp3.Highlighted phrase 3 looks like this.:ehp3.
:li.:hp4.Highlighted phrase 4 looks like this.:ehp4. (BLUE)
:li.:hp5.Highlighted phrase 5 looks like this.:ehp5.
:li.:hp6.Highlighted phrase 6 looks like this.:ehp6.
:li.:hp7.Highlighted phrase 7 looks like this.:ehp7.
:li.:hp8.Highlighted phrase 8 looks like this.:ehp8. (RED)
:li.:hp9.Highlighted phrase 9 looks like this.:ehp9. (PINK)
:esl.
```

The following example shows the output produced by these tags.

Formatted Output

Highlighted phrase 1 looks like this.

Highlighted phrase 2 looks like this.

Highlighted phrase 3 looks like this.

Highlighted phrase 4 looks like this.

Highlighted phrase 5 looks like this.

Highlighted phrase 6 looks like this.

Highlighted phrase 7 looks like this.

Highlighted phrase 8 looks like this.

Highlighted phrase 9 looks like this.

Highlighted Phrases

The type styles displayed for highlighted phrases correspond to the typeface currently being used by IPF. You can change the typeface to Courier, Helvetica** or Times New Roman** by using the :font. tag. See [Changing Fonts](#).

** Helvetica is a trademark of Linotype AG

** Times New Roman is a trademark of Monotype Corporation

Notes

To include notes in your information, you use a note tag: either :note. or :nt. (with its corresponding :ent.).

The one you use depends on whether your note consists of one paragraph or more than one.

:note.

Use :note. for single-paragraph notes. You do not need an end tag.

Following is an example of :note. and the resulting output.

Input Example

`:note.`Complete all entry fields before leaving
this window. If you do not, all your information will be lost.

Formatted Output

Note: Complete all entry fields before leaving this window. If you do not, all your information will be lost.

`:nt.`

Use `:nt.` to create notes with more than one paragraph. Remember to end the note with `:ent.`. In the following example, notice how the IPF compiler indents the text for the paragraphs in the note.

Input Example

```
:nt.Complete all entry fields before leaving  
this window. If you do not, all your information will be lost.  
:p.If your information is lost, retype it in  
the entry fields.  
:ent.
```

Formatted Output

Note: Complete all entry fields before leaving this window. If you do not, all your information will be lost.

If your information is lost, retype it in the entry fields.

Another Name for a Note: Both `:nt.` and `:note.` provide the `text=` attribute, so you can substitute your own word or phrase for the word "Note." The following shows the use of this attribute:

Input Example

```
:note text='Reminder'.Complete all  
entry fields before leaving this window.
```

Formatted Output

Reminder: Complete all entry fields before leaving this window.

Notices

Two tags enable you to include caution and warning notices in your information. Both tags require end tags.

`:caution.`

The following example shows how `:caution.` is used:

Input Example

```
:caution.  
Be sure to save your data. If you do not, all data will be lost.  
:ecaution.
```

Formatted Output

CAUTION:

Be sure to save your data. If you do not, all data will be lost.

:warning.

The following example shows how **:warning.** is used:

Input Example

```
:warning.  
The disk contains bad sectors.  
:ewarning.
```

Formatted Output

Warning: The disk contains bad sectors.

Place the caution and warning statements before the help information to which they apply so the user is cautioned or warned in advance. You can use the **text=** attribute if you want to use words other than "Caution" and "Warning" with these notices.

Simple List

Simple lists are vertical arrangements of items without any symbol or character preceding the items in the list. Use simple lists when the order of the items is not important.

To create a simple list, use the simple-list tag (**:sl.**) to begin the list and its corresponding end tag, **:esl.**. Identify each item in the list with a list-item tag (**:li.**).

Input Example

```
:p.Bring the following for lunch:  
:sl.  
:li.Fruit  
:li.Sandwich  
:li.Drink  
:esl.
```

Formatted Output

Bring the following for lunch:

Fruit
Sandwich
Drink

A Compact Simple List: Use the **compact** attribute to produce a list with no blank lines between the list items.

Input Example

```
:p.Bring the following for lunch:  
:sl compact.  
:li.Fruit  
:li.Sandwich  
:li.Drink  
:esl.
```

Formatted Output

Bring the following for lunch:

Fruit
Sandwich
Drink

Nested Lists: A nested list is a list that is contained within another list. The following shows the tagging for a simple list nested within another simple list, and the resulting output.

Input Example

```
:p.Bring the following for lunch:  
:sl.  
:li.Fruit, for example:  
:sl compact.  
:li.Apple  
:li.Orange  
:li.Pear  
:li.Banana  
:esl.  
:li.Sandwich  
:li.A drink  
:esl.
```

Formatted Output

Bring the following for lunch:

Fruit, for example:

Apple
Orange
Pear
Banana

Sandwich

A drink

Unordered List

Unordered lists are vertical arrangements of items, with each item in the list preceded by a special character, usually the lowercase "o" (called a *bullet*).

Use unordered lists when the order of the items is not important.

To create an unordered list, use the unordered-list tag (:ul.) to begin the list and :eul. to end it. Identify each item in the list with :li..

Input Example

```
:ul.  
:li.Information typed in Window A will be stored in the  
STORES.DAT file in whatever directory you designate.  
:li.Information typed in Window B will be stored in the  
SALES.DAT file in the current directory.  
:li.Information typed in Window C will be stored in the  
LOSSES.DAT file in the C:\FINANCE directory.  
:eul.
```

Formatted Output

- Information typed in Window A will be stored in the STORES.DAT file in whatever directory you designate.
- Information typed in Window B will be stored in the SALES.DAT file in the current directory.
- Information typed in Window C will be stored in the LOSSES.DAT file in the C:\FINANCE directory.

Note: To change bullet or dash character, change the appropriate .NLS file coding of ULITEM IDX=' ', located in the \TOOLKIT\IPFC subdirectory.

A Compact Unordered List: Use the **compact** attribute to produce a list with no blank lines between the list items.

Input Example

```
:ul compact.
:li.Information typed in Window A will be stored in the
STORES.DAT file in whatever directory you designate.
:li.Information typed in Window B will be stored in the
SALES.DAT file in the current directory.
:li.Information typed in Window C will be stored in the
LOSSES.DAT file in the C:\FINANCE directory.
:eul.
```

Formatted Output

- Information typed in Window A will be stored in the STORES.DAT file in whatever directory you designate.
- Information typed in Window B will be stored in the SALES.DAT file in the current directory.
- Information typed in Window C will be stored in the LOSSES.DAT file in the C:\FINANCE directory.

Nested Unordered Lists: The following example contains two nested, unordered lists. Notice that a bullet (lowercase "o") precedes items in the first-level list and that a dash (-) precedes items in the second-level lists. The bullets and dashes alternate for each level of the list. That is, third-level list items would be preceded by bullets, fourth-level by dashes, and so on.

Input Example

```
:ul compact.
:li.C:\REPORTS\SALES.89
:ul compact.
:li.FIRST.QTR
:li.SECOND.QTR
:li.THIRD.QTR
:li.FOURTH.QTR
:eul.
:li.C:\REPORTS\SALES.90
:ul compact.
:li.FIRST.QTR
:li.SECOND.QTR
:li.THIRD.QTR
:li.FOURTH.QTR
:eul.
:eul.
```

Formatted Output

- C:\REPORTS\SALES.89
 - FIRST.QTR
 - SECOND.QTR
 - THIRD.QTR
 - FOURTH.QTR
- C:\REPORTS\SALES.90
 - FIRST.QTR
 - SECOND.QTR
 - THIRD.QTR
 - FOURTH.QTR

When nesting lists, make sure you end each list with an end-list tag.

Ordered List

Ordered lists are vertical arrangements of items, with each item in the list preceded by a number or letter. Use ordered lists when the sequence of the items is important, such as in a procedure.

To create an ordered list, use the ordered-list tag (**:ol.**) to begin the list and **:eol.** to end it. Identify each item in the list with **:li.**

Input Example

```
:ol.
:li.Open the diskette-drive door.
```

```
:li.Remove the diskette.  
:li.Store the diskette in a safe place.  
:eol.
```

Formatted Output

1. Open the diskette-drive door.
2. Remove the diskette.
3. Store the diskette in a safe place.

A Compact Ordered List: Use the **compact** attribute to produce a list with no blank lines between the list items.

Input Example

```
:ol compact.  
:li.Open the diskette-drive door.  
:li.Remove the diskette.  
:li.Store the diskette in a safe place.  
:eol.
```

Formatted Output

1. Open the diskette-drive door.
2. Remove the diskette.
3. Store the diskette in a safe place.

Nested Ordered Lists: The following example contains two nested, ordered lists. Notice that sequential numbers precede items in the first-level list, and sequential letters precede items in the second-level list. Numbers and letters alternate for each level of the list. That is, third-level list items would be preceded by numbers, fourth-level by letters, and so on.

Input Example

```
:ol.  
:li.First item in the first-level list.  
:li.Second item in the first-level list.  
This item has a nested list within it.  
:ol.  
:li.First item in the second-level list.  
:li.Second Item in the second-level list.  
:eol.  
:li.Third item in the first-level list.  
:eol.
```

Formatted Output

1. First item in the first-level list.
2. Second item in the first-level list. This item has a nested list within it.
 - a. First item in the second-level list.
 - b. Second Item in the second-level list.
3. Third item in the first-level list.

When nesting lists, make sure you end each list with an end-list tag.

Definition List

A definition list is a special list that pairs a term and its description.

To create a definition list, use the definition-list tag (**:dl.**) to begin the list and **:edl.** to end it. Identify each term in the list with a definition-term tag (**:dt.**) and each description with a definition-description tag (**:dd.**).

Column Width for Definition Terms: **:dl.** has several attributes that let you control the appearance of definition lists. The **tsize=** attribute specifies the width, in character spaces, for the term column. If **tsize=** is not specified, the default width for the term column is 10 character

spaces.

Definition-List Headings: If you want headings for the columns of terms and definitions, use the definition-term heading tag (`:dthd.`) to identify the heading for the terms and the definition-description tag (`:ddhd.`) to identify the heading for the definition descriptions.

Compact Definition List: The **compact** attribute produces a list with no blank lines.

The following example shows the tagging for a compact definition list with headings for the terms and descriptions. It also shows the use of the **tsize=** attribute.

Input Example

```
:dl compact tsize=13.
:dthd.:hp2.Key:ehp2.
:ddhd.:hp2.Purpose:ehp2.
:dt.Insert key
:dd.Switches between insert and replace modes.
:dt.Home key
:dd.Moves the cursor to the beginning of the current line.
:dt.End key
:dd.Moves the cursor to the end of the current line.
:ed1.
```

Formatted Output

Key	Purpose
Insert key	Switches between insert and replace modes.
Home key	Moves the cursor to the beginning of the current line.
End key	Moves the cursor to the end of the current line.

Specifying where the Definition Descriptions Start: The **break=** attribute defines where the descriptions appear in relation to their terms:

break=none Places the description on the same line as the term. This is the default. If the term is longer than the specified or default **tsize=** value, the term extends into the description column.

break=all Places the description on the line below the term.

break=fit Places the description on the line below the term only when the term is longer than the **tsize=** value.

The following example shows the tagging that starts the definition descriptions on the line below the term.

Input Example

```
:dl break=all tsize=3.
:dt.:hp2.Insert key:ehp2.
:dd.Switches between insert and replace modes.
:dt.:hp2.Home key:ehp2.
:dd.Moves the cursor to the beginning of the current line.
:dt.:hp2.End key:ehp2.
:dd.Moves the cursor to the end of the current line.
:ed1.
```

Formatted Output

Insert key	Switches between insert and replace modes.
Home key	Moves the cursor to the beginning of the current line.
End key	Moves the cursor to the end of the current line.

A definition description can apply to more than one definition term; that is, you can specify more than one `:dt.` in the sequence before specifying a matching `:dd.`

The following example shows the tagging for a definition list with descriptions that apply to more than one term.

Input Example

```
:dl compact break=fit tsize=20.
:dthd.:hp2.Grocery Item:ehp2.
:ddhd.:hp2.Type:ehp2.
```

```
:dt.:hp2.Orange:ehp2.
:dt.:hp2.Apple:ehp2.
:dd.A fruit.
:dt.:hp2.Carrot:ehp2.
:dt.:hp2.Celery:ehp2.
:dd.A vegetable.
:ed1.
```

Formatted Output

Grocery Item	Type
Orange	
Apple	A fruit.
Carrot	
Celery	A vegetable.

Parameter List

Parameter lists are similar to definition lists in appearance and the way you use tags to create them. The only difference between the two types of lists is that a parameter list cannot have headings.

The parameter-list tag (**:parml.**) begins the list; its corresponding **:eparml.** ends it. Identify each term in the list with a parameter-term tag (**:pt.**) and each description with a parameter-description tag (**:pd.**).

:parml. has the same attributes as **:dl.** The **tsize=** attribute specifies the width for the term column. If **tsize=** is not specified, the default width is 10 character spaces.

Compact Parameter List: The **compact** attribute produces a list with no blank lines.

Specifying where the Parameter Descriptions Start: The **break=** attribute defines where the descriptions appear in relation to their terms:

break=none	Places the description on the same line as the term. This is the default. If the term is longer than the specified or default tsize= value, the term extends into the description column.
break=all	Places the description on the line below the term.
break=fit	Places the description on the line below the term only when the term is longer than the tsize= value.

Nested Parameter Lists: Like simple, unordered, and ordered lists, parameter lists can be nested.

Input Example

```
:parml compact tsize=3.
:pt.:hp2.KEYWORD-1:ehp2.
:pd.Is explained here.
:pt.:hp2.KEYWORD-2:ehp2.
:pd.Is explained here, and its nested subparameters:
:parml compact.
:pt.:hp2.SUBPARAM1:ehp2.
:pt.:hp2.SUBPARAM2:ehp2.
:pd.Are explained here.
:eparml.
:pt.:hp2.KEYWORD-3:ehp2.
:pd.Is explained here.
:eparml.
```

Formatted Output

KEYWORD-1	
Is explained here.	
KEYWORD-2	
Is explained here, and its nested subparameters:	
SUBPARAM1	
SUBPARAM2	
	Are explained here.
KEYWORD-3	
Is explained here.	

A parameter description can apply to more than one parameter; that is, you can specify more than one **:pt.** in the sequence before specifying a matching **:pd.**

The following example shows the tagging for a parameter list with descriptions that apply to more than one term.

Input Example

```
:parml compact tsize=3.  
:pt.:hp2.KEYWORD-1:ehp2.  
:pt.:hp2.KEYWORD-2:ehp2.  
:pd.Is explained here.  
:pt.:hp2.KEYWORD-3:ehp2.  
:pd.Is not explained here.  
:eparml.
```

Formatted Output

```
KEYWORD-1  
KEYWORD-2  
    Is explained here.  
KEYWORD-3  
    Is not explained here.
```

Tables

Table tags enable you to display text in an arrangement of rows and columns. The system font used to create tables is the monospace font.

The table tag (**:table.**) signals the start of the table. It requires a corresponding **:etable.** at the end of the table.

The row tag (**:row.**) specifies the start of each row in the table. Each row must have at least one column-entry tag (**:c.**). This tag specifies the text for each column in the table.

The **cols=' '** attribute of **:table.** specifies numeric values that represent the column widths, in character spaces, of each column in the table. The combined values cannot exceed 250 characters.

The number of columns in your table is determined by the number of column width values you have specified with the **cols=' '** attribute. For example, if you enter the values shown in the following, your table will have three columns, each of which will be eleven characters spaces wide.

```
cols='11 11 11'
```

A Table with Three Columns: The following is a simple example of a table with two rows and three columns:

Input Example

```
:table cols='13 13 13'.  
:row.  
:c.Column 1  
:c.Column 2  
:c.Column 3  
:row.  
:c.Row 1 Col 1  
:c.Row 1 Col 2  
:c.Row 1 Col 3  
:row.  
:c.Row 2 Col 1  
:c.Row 2 Col 2  
:c.Row 2 Col 3  
:etable.
```

Formatted Output

```
Column 1      Column 2      Column 3  
  
Row 1 Col 1 Row 1 Col 2 Row 1 Col 3  
  
Row 2 Col 1 Row 2 Col 2 Row 2 Col 3
```

If you have more **:c.** tags following a **:row.** tag than you have column-width values, the extra column entries are placed in a new row, and the compiler returns an error message.

If you have fewer **:c.** tags than column-width values, the compiler does not consider this an error. Space is still allocated for the specified columns; however, only the columns for which you have provided entries will be filled.

Table Rules and Frames: The **rules=** attribute of **:table.** specifies whether the table will have vertical rules, horizontal rules, a combination of both, or no rules at all to delineate the items in the table. The values that you can specify for **rules=** are:

rules=both
rules=none
rules=horiz
rules=vert

A Table with Horizontal Rules:

Input Example

```
:table rules=horiz cols='10 15 15'.
:row.
:c.SYMBOL
:c.ELEMENT
:c.CHARACTER
:row.
:c.&amp;bxas.
:c.box ascender
:c.&bxas.
:row.
:c.&amp;bxcr.
:c.box cross
:c.&bxcr.
:row.
:c.&amp;bxde.
:c.box descender
:c.&bxde.
:etable.
```

Formatted Output

SYMBOL	ELEMENT	CHARACTER
&bxas.	box ascender	
&bxcr.	box cross	
&bxde.	box descender	

If you do not specify the **rules=** attribute, your table will contain both vertical and horizontal rules (the default).

The **frame=** attribute of **:table.** specifies whether the table will have borders. The values that you can specify are:

frame=none
frame=rules
frame=box

If you specify **frame=none**, there will be no borders.

Specifying **frame=rules** results in a horizontal line at the top and bottom of the table.

If you specify **frame=box**, or do not specify the **frame=** attribute, the table is enclosed in a box.

A Table without a Frame: Here is the same table without a frame.

Input Example

```
:table rules=horiz frame=none cols='10 15 15'.
:row.
:c.SYMBOL
```

```

:c.ELEMENT
:c.CHARACTER
:row.
:c.&amp;.bxas.
:c.box ascender
:c.&bxas.
:row.
:c.&amp;.bxcr.
:c.box cross
:c.&bxcr.
:row.
:c.&amp;.bxde.
:c.box descender
:c.&bxde.
:etable.

```

Formatted Output

SYMBOL	ELEMENT	CHARACTER
&bxas.	box ascender	"
&bxcr.	box cross	
&bxde.	box descender	"

Special Considerations: None of the text-formatting tags (for example, list tags) can be used in a table. You can use character-graphic symbols and highlighted-phrase tags. However, boldface and italic highlighting can cause vertical misalignment of column text and rules. To use boldface highlighting in tables and avoid word alignment problems, place the highlighted-phrase tags (**:hp2.** and **:ehp2.**) as shown in the example. The table rules as well as the text will be displayed in boldface.

Input Example

```

:hp2.
:table cols='11 11 11'.
:row.
:c.Row 1 Col 1
:c.Row 1 Col 2
:c.Row 1 Col 3
:row.
:c.Row 2 Col 1
:c.Row 2 Col 2
:c.Row 2 Col 3
:etable.
:ehp2.

```

Formatted Output

Row 1 Col 1	Row 1 Col 2	Row 1 Col 3
Row 2 Col 1	Row 2 Col 2	Row 2 Col 3

The above technique is practical only with **:hp2.** and does not work for **:hp1.** or for the highlighted-phrase tags that change the color of text.

Unformatted Text in a Table Column: The text in table columns is formatted only once-at compile time. If you do not want the compiler to format the text in a column, enclose it with **:lines.** and **:elines.**, as shown in the following example.

Input Example

```

:table cols='10 10 15 10'.
:row.
:c.Spacecraft
:c.Date
:c.Astronauts
:c.Mission

```

```

:row.
:c.Apollo 11
:c.7-16-1969
:c.
:lines.
Neil Armstrong
Edwin Aldrin
Michael Collins
:elines.
:c.
First landing on the moon.
:etable.

```

Formatted Output

Spacecraft	Date	Astronauts	Mission
Apollo 11	7-16-1969	Neil Armstrong Edwin Aldrin Michael Collins	First landing on the moon.

Plain Lines

Normally, lines of text that have no formatting tags are wrapped by IPF; that is, irregular lines in the source file become a continuous string, and one word follows another on a line until the line width of the current window is filled, a formatting tag is encountered, or the end of the window is reached.

The **:lines.** tag and its corresponding end tag, **:elines.**, enable you to control where lines break.

The following example shows how the **:lines.** tag prevents wrapping of text.

Input Example

Here is how IPF normally wraps lines to fit the window width:

```

:p.&odq.Normally, lines of text that have no formatting tags are wrapped by IPF; that is, irregular lines in the source file become a continuous string, and one word follows another on a line until the line width of the current window is filled, a formatting tag is encountered, or the end of the window is reached.&cdq.

```

In the following example, IPF will not wrap the lines, because we used the **:hp2.&colon.lines&per.:ehp2.** tag to prevent the lines from being formatted.

```

:lines.
&odq.Normally, lines of text that have no
formatting tags are wrapped by IPF;
that is, irregular lines in the source
file become a continuous string, and
one word follows another on a line
until the line width of the current
window is filled, a formatting tag
is encountered, or the end of the
window is reached.&cdq.
:elines.

```

The quotation appears in two forms.

Plain Lines Example

Here is how IPF normally wraps lines to fit the window width:

"Normally, lines of text that have no formatting tags are wrapped by IPF; that is, irregular lines in the source file become a continuous string, and one word follows another on a line until the line width of the current window is filled, a formatting tag is encountered, or the end of the window is reached."

In the following example, IPF will not wrap the lines, because we used the **:lines.** tag to prevent the lines from being formatted.

"Normally, lines of text that have no

formatting tags are wrapped by IPF; that is, irregular lines in the source file become a continuous string, and one word follows another on a line until the line width of the current window is filled, a formatting tag is encountered, or the end of the window is reached."

In the first case, IPF wraps the lines to fit the window width. In the second, IPF does not wrap the lines, because **:lines.** prevents them from being formatted. If a line of text were to exceed the width of the current window, it would be clipped. Also, when IPF encounters other tags between **:lines.** and **:elines.**, such as quotation tags, the tags are processed.

Aligned Lines: **:lines.** has an **align=** attribute, which you use to align text to the left, right, or center of the window.

Assume that in the previous example, the tag was:

```
:lines align=center.
```

The output would be as shown here.

Normally, lines of text that have no formatting tags are "wrapped" by IPF; that is, irregular lines in the source file become a continuous string, and one word follows another on a line until the line width of the current window is filled, a formatting tag is encountered, or the end of the window is reached.

Text aligned in the center of the window.

Figures and Captions

The figure tag (**:fig.**) is similar to **:lines.**. Both convey the same message: "Do not format the text that follows." Also, both tags have end tags.

A Captioned Figure: Associated with **:fig.** is **:figcap.**, which enables you to place a descriptive sentence or caption above or below the text.

Input Example

```
:h4.Example 1: A Captioned Figure  
:fig.  
  
Bat  
Black Bear  
Bobcat  
Coyote  
Mink  
Florida Panther  
Key Deer  
Opposum  
West Indian Manatee  
Whitetail Deer  
  
:figcap.Major Species of Mammals in Florida  
:efig.
```

The formatted output looks like this:

Example 1: A Captioned Figure

```
Bat  
Black Bear  
Bobcat  
Coyote  
Mink  
Florida Panther
```

Key Deer
Oppossum
West Indian Manatee
Whitetail Deer

Major Species of Mammals in Florida

Figure and Figure Caption

Textual Examples

One way of helping readers understand information is to use examples. The example tag (**:xmp.**) and its corresponding end tag (**:exmp.**) enable you to illustrate your information with textual examples by turning formatting off so that you can arrange text any way you want it. The text will be displayed in a monospace font. To change the monospace font, use **:font.** within **:xmp.** For more information about **:font.**, see [Changing Fonts](#).

Input Example

```
:xmp.
File   Edit   View   Options   Help

      All
      Some . . .
      By . . .

:exmp.
```

Formatted Output

```
File   Edit   View   Options   Help

      All
      Some . . .
      By . . .
```

Restriction: You cannot nest **:xmp.** within another **:xmp.**

Character Graphics

If you want to include simple line drawings, use the character graphics tag (**:cgraphic.**) and its corresponding end tag (**:ecgraphic.**). Text within this tag is displayed in a monospace font. To change the monospace font, use **:font.** within **:cgraphic.** For more information about **:font.**, see [Changing Fonts](#). If text does not fit within the boundaries of a window, it is clipped, not wrapped.

Place the tags before and after the character graphic, as shown in the following example.

Input Example

```
:cgraphic.

File   Edit   View   Options   Help

      All
      Some . . .

      By . . .

:ecgraphic.
```


Formatted Output

```
File   Edit   View   Options   Help

      All
      Some . . .

      By . . .
```

Restriction: You cannot nest **:cgraphic.** within another **:cgraphic.**.

Changing Fonts

The **:font.** tag is used to change the current font within the text of the current window. When a heading tag that defines a new window is encountered, the font is reset to the system default font.

The font tag has three attributes: **facename=** and **size=** are required; **codepage=** is optional. If a code page value is not specified, the code page of the active system is used.

facename= specifies the name of the font you want to change to. Some of the common values for this attribute are:

```
Helv
Courier
default
```

size= specifies the height and width, in *points*, of the font you have selected. (A *point* is a typesetting measure equal to approximately 1/72 of an inch.) The value is expressed in the form, *HxW*. For example, suppose you want to change the current font to an 18-point-high by 10-point-wide Helvetica font. You would specify:

```
:font facename=Helv size=18x10.
```

You do not have to know exact point values. IPF uses a "best fit" method to select the font. If, in the example above, you had specified *20x12* as the size value, IPF would have selected *Helv 18x10* because it is the closest size to the one you specified.

Using **:font.**, you can make as many font changes within a window as you want. You can define highlighted phrases while a font tag is in effect, and the tagged text will be displayed in the font style corresponding to that typeface.

You can use **:font.** within the **:xmp.** and **:cgraphic.** tags to change the default system monospace font. To change the default system monospace font, specify the desired **facename=** and **size=** attribute.

The following resets the font to the default system proportional font.

```
:font facename=default size=0x0.
```

In the following example, the font style is reset for each list item in the simple list.

Input Example

```
:p.The following illustrate available fonts:
:sl.
:font facename=Courier size=13x8.
:li.This sentence is in Courier 13 by 8 font.
.*
:font facename='Tms Rmn' size=18x14.
:li.This sentence is in 'Tms Rmn' 18 by 14 font.
.*
:font facename=Helv size=28x18.
:li.This sentence is in Helvetica 28 by 18 font.
.*
:font facename=default size=0x0.
:li.This sentence is in the default system font.
:esl.
```

Here is the formatted output.

Example of the Font Tag

This sentence is in Courier 13 by 8 font.

This sentence is in 'Tms Rmn' 18 by 14 font.

This sentence is in Helvetica 28 by 18 font.

This sentence is in the default system font.

Changing Color

The color tag (**:color.**), with its attributes **fc=** and **bc=**, enables you to change the color of the text (foreground color) and the color of the area behind the text characters (background color).

Colors set with this tag remain in effect until others are specified, or until a heading definition is encountered.

To return to the system colors, specify:

```
:color fc=default bc=default.
```

In the following example, each of the first three color tags specifies different foreground and background colors. The last color tag returns the colors to the system colors.

Input Example

```
:ol.  
:color fc=green bc=blue.  
:li.Color the foreground green; color the background blue.  
.*  
:color fc=blue bc=red.  
:li.Color the foreground blue; color the background red.  
.*  
:color fc=cyan bc=yellow.  
:li.Color the foreground cyan; color the background yellow.  
.*  
:color fc=default bc=default.  
:li.Return to the system colors.  
:eol.
```

Output Example: Color the Foreground and Background

Color the foreground green, color the background blue

Color the foreground blue, color the background red

Color the foreground cyan, color the background yellow.

Return to the system colors.

Margins

You can specify the boundaries of text in a window by using the margin tags. The left-margin tag (**:lm.**) specifies how many character spaces

from the left border of the window the text is to start. The right-margin tag (:rm.) specifies how many character spaces from the right border the text is to end.

The **margin=** attribute sets the margin for the text. If none is specified on the :lm. or :rm. tag, the default is 1.

If the margin tag in a line begins beyond the specified boundary, the new margin becomes effective on the next line.

You can have multiple margin tags in your file. The specified margins remain in effect until they are reset.

Input Example

```
:p.  
:rm margin=10.  
:lm margin=20.This text begins 20 spaces to the right  
of the left window border and ends 10 spaces to the  
left of the right window border.  
All text is aligned as specified  
by the margin values. :lm margin=5.Here the left margin  
is changed to 5. Because this margin tag begins  
more than 5 spaces on the line, the margin specified  
becomes effective on the following line, and the text  
begins 5 spaces from the left window border.  
The right margin remains unchanged.
```

Here is how the window looks:

This text begins 20 spaces to the right of the left window border and ends 10 spaces to the left of
the right window border. All text is aligned as specified by the margin values. Here the left margin
is changed to 5. Because this margin tag begins more than 5 spaces on the line, the margin specified becomes effective on the
following line, and the text begins 5 spaces from the left window border. The right margin remains unchanged.

Example of the Margin Tag

Bit Map and Metafile Graphics

In a previous topic, we discussed how you can use :cgraphic. to illustrate your text with character graphics. With :artwork., you can illustrate your text with *bit-map* or *metafile* graphics. A bit map is a representation of an image, and can be created with such tools as the Icon Editor, which is available with the *IBM Developer*. Metafiles provide device independence; bit maps do not. The bit map or metafile graphics reside in a file that must be specified with the **name=** ' ' attribute of :artwork.. This file is then loaded when you compile your source file with the IPF compiler. The OS/2 2.0 compress bit map format is not supported by the IPF compiler. The IPF compiler compresses the bit maps.

The artwork tag has other attributes as well:

- The **align=** attribute enables you to position the graphic. The values are **left**, **right**, and **center**, and are with respect to the current margins.
- The **fit** attribute causes a bit map to be redrawn and scaled to fit the window.


The ratio between the width and height of the window should be the ratio of the original width and height of the bit map or metafile; otherwise, the graphic might appear distorted.
- The **runin** attribute enables you to place a graphic within a line of text. For example, to include an icon within a line of text, the text and tag would be as follows:

Input Example

```
:p.This is an example of artwork displayed within the  
:artwork runin name='BOOK.BMP'.  
text of a sentence.  
.**  
:p.You can also align the artwork to appear on the  
:lines align=left.  
left,  
:elines.  
:artwork align=left name='BOOK.BMP'.  
:lines align=right.  
right,  
:elines.  
:artwork align=right name='BOOK.BMP'.
```

```
:lines align=center.
or center of the window.
:elines.
:artwork align=center name='BOOK.BMP'.
```

It would bring the artwork into the screen like this.

This is an example of artwork displayed within the  text of a sentence.

You also can align the artwork to appear on the left,



right,



or center of the window.



Example of the Artwork Tag

Linking

Today, the computer's ability to link pieces of information gives the author flexibility in layering and structuring documents, and at the same time, provides cohesive information.

This chapter describes the tags that identify, associate, and link one window to another window. This chapter also describes the different types of linking available with IPF, and what to expect when using them.

Window Identifiers

The link tag (:link.) allows you to link to a heading, a footnote, an external database, or another application. The **reftype=** attribute is required with each link tag description. This attribute identifies the type of link you are defining.

The **res=** attribute and the value specified identify the window you are linking to. This attribute is the window identifier. A **res=** number must be in the range 1 through 64 000. The same window identifier must be specified in the tagging of the window you are linking to in order for a hypertext link to exist (see [Hypertext Links](#)).

The IPF compiler recognizes links to headings (including hidden headings) only when the heading level is within the default range (**toc=123**) or specified range of heading levels. If you specify a window identifier for a level that is lower in the hierarchy than that recognized for contents entries, and then attempt to link to it, the compiler returns an error message. For example, suppose the default is in effect for contents entries; that is, only heading levels 1 through 3 cause entries in the Contents window. Also suppose your file contains the following heading definition:

```
:h4 res=050.Copy File
```

The heading "Copy File" appears in the same window as the preceding heading level 3. If you use this window identifier in a link definition to link to the heading from another window, the IPF compiler returns the error message, `No res for this reference.`

If you are creating windows for an online document (a .INF file), you can use the **res=**, **id=**, or **name=** attribute of the heading tag to specify window identifiers. An advantage of using either **id=** or **name=** is that you can specify both alphabetic and numeric characters, which can make the job of assigning and remembering window IDs easier. If you use one of these attributes, you must use the **refid=** attribute of **:link.** when defining a hypertext cross-reference to the window.

If you need to use both **res=** numbers and **id=** values, you can specify both in a window heading. For simplicity, you can assign the same number to both identifiers.

Note: If an OS/2 application needs to communicate with an IPF window, you must use the **res=** attribute as a window identifier.

Types of Links

Links are electronic connections between one online element and another. With IPF, the user can be linked from one window to another by means of selectable text and graphic areas that the author defines. The user also can be linked to information in another IPF database.

Different types of links support document designs and information retrievability in various ways:

Hypertext Links Selectable words or phrases that connect related information.

Hypergraphic Links
 Selectable graphics that connect related information.

Automatic Links Links that begin a chain reaction at the primary window. When the user selects the primary window, an automatic link is activated to display secondary windows.

External Links Links that connect external online document files.

Hypertext Links

Hypertext is the linking of online information so the user can navigate from selectable text to related information. A hypertext link is the association between two topics. The *origin* of the link is the source topic; the *destination* is the target topic.

In the following example, the DIR command is the source topic; it describes the directory command. Within the DIR topic is a reference to the MKDIR command - the target topic.

Source Topic	Target Topic
DIR - Display files in ...	MKDIR - Make a new ...
Related command: MKDIR	Related command: DIR

You use **:link** to establish a hypertext link between a topic in the source-topic window and a topic in the target-topic window. **:link** enables you to create selectable, highlighted text in the source-topic window. When users select this text, they are linked to the window containing the target topic, and the linked window appears.

Consider the following example:

```
:link reftype=hd res=123.MKDIR:elink.
```

- **reftype=hd** indicates the hypertext phrase *MKDIR* is being linked to a heading in the target-topic window.

Notice MKDIR is delimited by the period of the **:link** tag and the colon of the **:elink** tag.

- **res=123** is the identifier of the target-topic window.

The heading tag of the target-topic window must contain this identifier. The following is an example:

```
:h2 res=123.MKDIR
```

For more information about hypertext links, see [Display Another Window of the Same Library](#).

Hypergraphic Links

A hypergraphic link is similar to a hypertext link except that the user navigates from a selectable graphic instead of selectable text.

Bit Maps

Graphic illustrations are usually bit maps. Bit maps can be monochrome or color and can be created with the Presentation Manager Icon Editor, which is available in the *IBM Developer*. The bit map resides in a separate file called by IPF at compile time.

The artwork tag (**:artwork.**) identifies the name of the bit-map; for example:

```
:artwork name='mybitmap.bmp'.
```

The **:artlink.** and **:eartlink.** tags define areas of the bit map that are selectable *hypergraphic*. This means the user can link from the artwork to additional information. If no **:artlink.** tag is used, no hypergraphic areas are defined.

If you want the entire bit map to be hypergraphic, the tagging is simple. You have only one art link, and you do not have to define the area. The following shows the tagging required to establish a link:

Input Example

```
:p.This is an example of a hypergraphic.  
Select the Shuttle graphic and get ready for a walk on  
the moon.  
:artwork name='shuttle.bmp'.  
:artlink.  
:link reftype=hd res=001.  
:eartlink.
```

Notice there is no **:elink.** tag. Instead, there is an **:eartlink.** tag. An **:elink.** tag is required only to denote the end of a hypertext link.

You also need to specify the identifier in the tagging for the window you are linking to. For example:

```
:h1 res=001.Apollo 11
```



The entire bit map as a selectable hypergraphic area.

When the user double-clicks on the hypergraphic area, the window whose identifier is 001 ("Apollo 11") appears.

Metafiles

A *metafile* is another type of file in which graphics are stored. However, a metafile contains data generated from the Presentation Manager graphics (GPI) functions only. (For information about graphics functions, see the OS/2 2.0 *Programming Guide*, Volume 3.) IPF supports a metafile as a hypergraphic link only when the entire metafile is defined as a hypergraphic area.

The artwork tag identifies the file name of a metafile as follows:

```
:artwork name='myfile.met'.
```

Segmented Hypergraphics

You can divide your bit map into rectangular segments, make each segment selectable, and have each segment link to different information. You must define each segment in terms of values along the x and y axes. Values for x and y define the origin of the segment. The changes in x and y are given as values for cx and cy. The following is an example of a segmented bit map:

```
0,16                                32,16
|
|
|
y
|
|
|
0,0 -----x----- 32,0
```

The following shows the tagging to establish a bit-map segment as a hypergraphic area:

```
:artwork name='show2.bmp'.
:artlink.
:link reftype=hd res=001 x=0 y=0 cx=16 cy=8.
:eartlink.
```

Automatic Links

Links also can be made automatically. An automatic link occurs when the user performs an action that selects a window in which a link is defined. For automatic links to occur, the **reftype=** attribute of the **:link.** tag must have a value of **hd**, **inform**, or **launch**. Automatic links allow you to:

- Display multiple windows when a heading or link definition is selected (**hd** attribute).
- Display multiple secondary windows within the coverage of a primary window (**hd** attribute).
- Send a message to the application when a window is displayed (**inform** attribute).
- Start a Presentation Manager program when a window is displayed (**launch** attribute).

Automatic links can be associated with selectable links so that another action occurs in addition to the display of a linked window. For example, a Presentation Manager program can be started, or a message can be sent to the application program.

Restriction: Linking automatically to an external database is not possible.

External Links

An external link is a link from a .HLP file to another .HLP file or from a .INF file to another .INF file.

If you are linking from one internal database to another, use the **res=** attribute. If you want to allow external databases to link to a window in your file, the window heading must contain the **global** attribute, and you must use the **id=** attribute as a window identifier.

Restriction: Linking automatically to an external database is not possible.

For more information about external links, see [Display a Help Window from Another Help Library](#).

What Linking Can Do

You now know that **:link.** makes text phrases and hypergraphic areas within a window selectable. When the user selects a hypertext or hypergraphic area, the following occurs, depending on the content of the **:link.** tag:

- Another window of the same library is displayed.
 - Another window of a different library is displayed.
 - A footnote window is displayed.
 - A message is sent to the application program.
 - Another application is started.
-

Display Another Window of the Same Library

When you want the user to link to another window in the current library, use the **reftype=hd** attribute with **:link..** For example:

```
:link
reftype=hd res=21084.What Are Libraries For?
:elink.
```

The **hd** attribute tells the compiler to link to a heading in another window. The **res=** attribute value specifies the identification of the window being linked to.

The text "What Are Libraries For?" is uniquely highlighted in the window so that the user knows it is selectable. If the user selects it, the window containing the heading defined by **res= 21084** appears.

Note: The highlighting of a hypertext phrase is done with a color selected by IPF and should not be confused with highlighted-phrase tags, which are used to change the type font. (See [Highlighted Phrases](#) for an explanation of these tags.)

The tagging shown in the following figure contains an example of the link tag. Also included is the tagging for the window being linked to.

```
:*****
:* In the following source, the text of the window
:* contains a heading tag with a window
:* identifier, a paragraph tag, and a hypertext
:* link to another window.
:*****
:h1 res=21083.The Library Manager
:il.object code libraries
:p.
The Library Manager (LIB) lets you create and maintain
libraries of object code. A library is an organized
collection
```


of object code; that is, a library contains functions and data that are already assembled or compiled and ready for linking with your programs. See:

```
:link
reftype=hd res=21084.What Are Libraries For?
:elink.
:p.
LIB works with both DOS and OS/2 files.
:*****
:* The following contains a heading tag with a
:* window identifier that matches the link-tag
:* res= attribute above.
:* This file also contains an unordered list.
:*****
:h2 res=21084.What Are Libraries For?
:p.Programming libraries of object code are used:
:ul.
:li.To support high-level languages.
:p.Most compilers include libraries to perform standard
operations, such as input/output and floating-point mathematics.
:p.
When your program refers to a library routine, the
compiler and linker combine the library routine with your
program.
:li.To perform complex and specialized activities, such
as database management or advanced graphics.
:p.Compilers include libraries for specialized tasks. You
also can use a library from a third party software vendor.
:li.To support your own work.
:p.If you have created routines that you use with a
variety of programs, you might want to consolidate these routines
into a library. You then can link to one library object module
rather than to a large group of object files.
:eul.
```

Display a Window Linked to Another Database

You also can link a user to a window in another IPF .HLP or .INF file. You must specify the file name with the **database=** attribute. If the following were in the source file, selection of the hypertext link would cause the file, EDITOR.HLP to be loaded, and the window whose ID is 001 to be displayed.

```
:link reftype=hd database='editor.hlp' refid=001.
Editing Functions
:elink.
```

The heading definition in the other file must contain the **global** attribute. If the link to the file cannot be resolved, the hypertext phrase in the link is not highlighted. For example, if the .INF or .HLP file is not available, IPF will not highlight the linked phrase. If the .INF or .HLP file becomes available, IPF will dynamically highlight the phrase.

Display a Help Window from Another Help Library

If you are creating a window for a help library (a .HLP file), you must use the **res=** attribute to assign an identifier to each window. For example:

```
:hl res=2001 id=2001 global.
Help for Copy
```

IPF uses the value specified for **res=** (any integer from 1 through 64 000) to associate a window with a user's request for help on a field or window of the application. If you use the **res=** attribute in a heading tag, you must also use it in a link tag when defining a hypertext cross-reference to the window. For example:

```
:link reftype=hd res=2001.
Help for Copy
```

:elink.

Display a Footnote Window

A footnote window results when the user selects a hypertext phrase that is linked to a footnote tag (:fn.). The text between :fn. and :efn. is what appears in the footnote window. The following is an example of the tagging for the footnote text:

Input Example

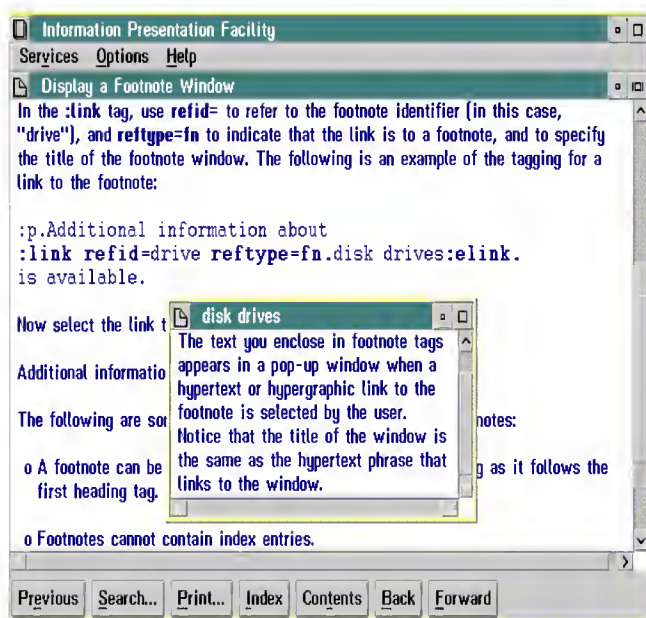
```
:fn id=drive.  
:p.The text you enclose in footnote tags appears in a  
small window when the user selects a hypertext or hypergraphic  
link to the footnote. Notice that  
the title of the window is the same as the hypertext  
phrase "disk drives" that links to the window.  
:efn.
```

The **id=** attribute identifies the footnote for linking purposes.

In the **:link** tag, use **refid=** to refer to the footnote identifier (in this case, "drive"), and **reftype=fn** to indicate that the link is to a footnote, and to specify the title of the footnote window. The following is an example of the tagging for a link to the footnote:

```
:p.Additional information about  
:link refid=drive reftype=fn.disk drives:elink.  
is available.
```

The following figure shows the resulting footnote window.



Footnote Window

The following are some important points to remember about footnotes:

- A footnote can be placed anywhere in your source file, as long as it follows the first heading tag.
- Footnotes cannot contain index entries.
- Information in a footnote cannot be detected by a search.

- A footnote cannot be in a window that has a **split** attribute in its heading or link definition.

Send a Message to the Application

When the **reftype=inform** attribute is specified with **:link.**, a message is sent to the application. The **res=** attribute, instead of being a resource identifier for IPF (a window ID), is a resource identifier for the application. The value specified must be an integer. When the application receives the message, it can then perform an application-specific function.

For more information about how messages are sent to application windows using the **reftype=inform** attribute see [Using Communication Windows](#).

Start an Application

The **reftype=launch** attribute of **:link.** causes IPF to start another Presentation Manager application. The **object=** attribute indicates the file specification of the application. The **data=** attribute specifies parameters associated with the application to be started.

You can use the **reftype=launch** attribute with **:link.** to start a tutorial.

Customizing Windows

A window is an area of the screen with visible boundaries within which information is displayed. Often a single window uses the entire screen for its information. Because online information is best presented in small pieces, or units, most designs call for a multiple window format. This chapter explains how to size and position more than one window on a screen, and how to use attributes that enable IPF to open and close those windows. Before you begin this chapter, make sure you read about the OS/2 standard windows described in [IPF User Interface](#).

For a summary of attributes described in this chapter, see [Summary Tables of Attribute Values for Origin and Size](#), [Summary Table for Heading Attributes](#), and [Summary Table for Link Attributes](#).

The Default Window

Both the heading tags (**:h_n.**) and the link tag (**:link.**) have attributes that affect how windows look on a screen. For example, the attributes define:

- Window size and position
- Which window controls are provided to the user
- What windows are displayed.

You do not have to use all the attributes provided by a heading tag to define a window. The following is an example of the minimum tagging required for a window:

```
:h1 res=001.My First Window
:p.
Here is the text for the first window.
```

In this example, **:h1.** creates a level-1 entry in the Contents window and the title, "My First Window," in the title bar of the default window.

The following figure shows the tagging to produce the two default windows shown in the example that follows the tagging.

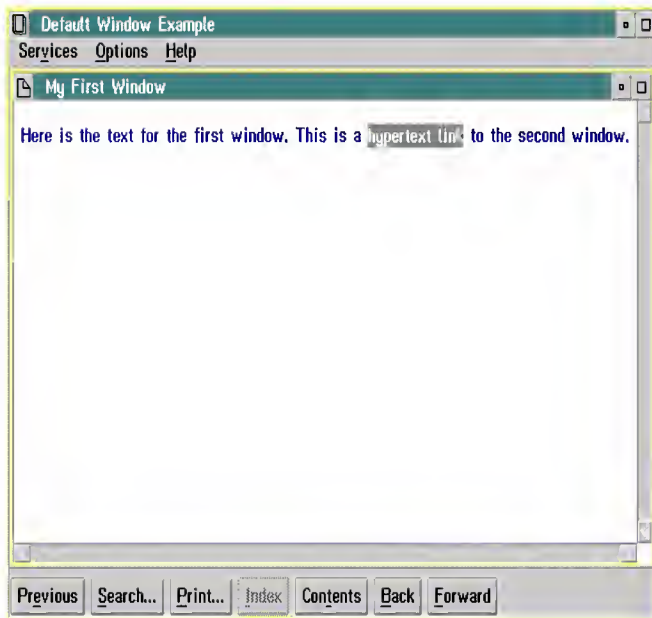
```
:h1 res=001.My First Window
:p.
Here is the text for the first window.
This is a
```

```

:link reftype=hd res=002.
hypertext link
:elink.
to the second window.
:h1 res=002.My Second Window
:p.
Here is the text for the second window.
This is a
:link reftype=hd res=001.
hypertext link
:elink. to the first window.

```

The following figure shows the compiled version of the tagging shown in the previous figure. "My First Window" is one of the default windows and is bounded by the window "Default Window Example." This window is called a *coverpage* and provides window controls for the user.



Example of an IPF Default Window

The two windows each have a hypertext link. Selection of the hypertext link in "My First Window" causes the other default window "My Second Window" to display. Each default window has the same characteristics:

- Its size is 100% of the coverpage window.
- It provides window controls for the user:
 - Title bar with a title bar icon
 - Maximize and hide buttons
 - Vertical and horizontal scroll bars
 - Sizing borders
 - Push buttons.

Attribute Values for Window Controls

Both the heading tag and `:link.` have attributes that define window controls. Following are the names of the window-control attributes, and values you can specify (defaults are underscored):

titlebar=yes|systemenu|minmax|both|none

scroll=horizontal|vertical|both|none

rules=border|sizeborder|none

You can eliminate window controls altogether by specifying:

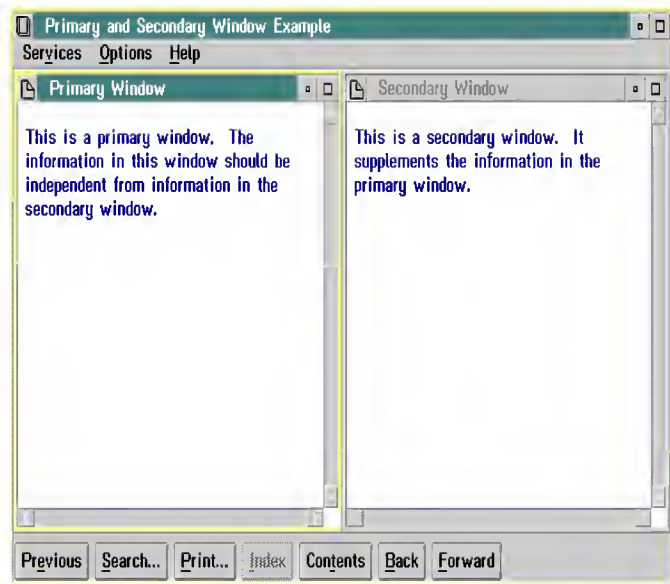
titlebar=none scroll=none rules=none

You then can substitute controls of your own. By eliminating borders around windows and using **:font.** to specify fonts, you can design a more sophisticated layout of text and graphics. The OS/2 system tutorial is an example of this.

For information about the tags that control the display of push buttons, see [Attribute Values for the Control Area of a Window](#).

Multiple Windows

Windows can be considered to be subdivisions of the screen. They can be either primary or secondary windows. A primary window is where the main topic appears, or where the interaction between a user and an object or application takes place. A secondary window usually supplements the information in the primary window. It is closed when its primary window is closed. The following figure shows a simple multiple-window design with a primary and secondary window.

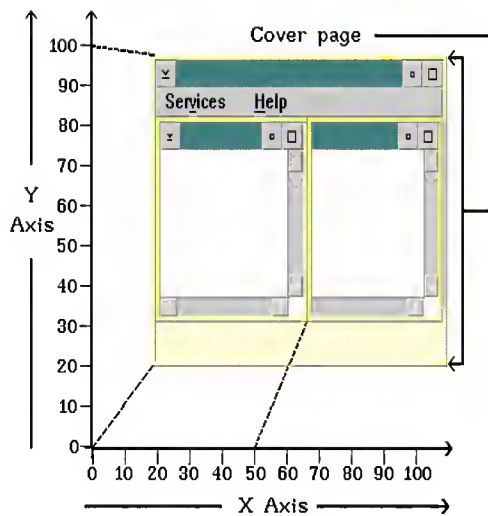


A Primary and Secondary Window Arrangement

To create the two-window format shown in the figure, you must define the size of each window, then position them within the boundaries of the coverage window. When defining window size, you specify horizontal and vertical areas of the window, using window coordinates.

Defining Window Origin and Size

Each window represents a rectangle with x and y coordinates. The x -axis is always horizontal; the y -axis is always vertical. The position where the values specified for x and y intersect is the window's origin. From this position, width and height are measured. The following figure shows the window coordinates of a primary and secondary window.



A Window in Relationship to Its Coordinates

Attribute Values for Window Origin and Size

Both the heading tags and `:link.` have attributes that define window origin and size. The heading tag has four attributes:

<code>x=</code>	Specifies a point on the x-axis. The x-axis runs horizontally from left to right.
<code>y=</code>	Specifies a point on the y-axis. The y-axis runs vertically from bottom to top.
<code>width=</code>	Specifies the width (horizontal space) of the window.
<code>height=</code>	Specifies the height (vertical space) of the window.

The `:link.` tag also has four attributes:

<code>vpX=</code>	Specifies a point on the x-axis. The x-axis runs horizontally from left to right.
<code>vpY=</code>	Specifies a point on the y-axis. The y-axis runs vertically from bottom to top.
<code>vpC=</code>	Specifies the width (horizontal space) of the window.
<code>vpCy=</code>	Specifies the height (vertical space) of the window.

Origin and size attributes also can be assigned values of the following types:

- Absolute
- Relative
- Dynamic

Absolute Values

Absolute values are specified in characters, pixels, or points. The format for an absolute value is an integer followed by one of these letters:

- c** (characters)
Average character width of the default system font.
- x** (pixels)
Pixel size, dependent on the display adapter in use.
- p** (points)
Typesetting measure, equal to approximately 1/72 inch.

Relative Values

Relative values are specified as percentages of the display area of the coverage window. The format for a relative value is an integer followed by the percent sign (%).

Dynamic Values

Dynamic values for x- and y-coordinates identify locations on the coverage-window perimeter or its center. Values are **left** and **right** for x, **top** and **bottom** for y, and **center** for both.

Heading Definition Example

The window defined in the following example is a primary window; its origin is specified using dynamic values, and its width and height are specified as percentages of its coverage window.

```
:h1 res=001
  x=left y=bottom width=50% height=100%
  group=1.Primary Window
```

For now, ignore "group=1." We will explain it later.

The most practical values to use for window size and position are a combination of relative and dynamic values. Then, if the user resizes the coverage window, IPF automatically resizes and repositions the windows relative to the new size and position of the coverage window. If you use absolute values, the window might be clipped when the user resizes the coverage window.

When defining window position and size, you cannot mix absolute values with dynamic or relative values for either of the following combinations of attributes:

x= and width=
y= and height=

If no values for x and y are specified, the origin of the window is 0,0. If you specify an origin other than 0,0, you also must specify width and height values. Negative values for these attributes are not allowed.

Origin and Size Example

The example of a source file shown in the following figure defines two windows. The origin and size attributes specified with the heading definitions place the windows adjacent to one another on the screen.

```
:h1.Origin and Size Window Example
:h2 res=003
  x=left y=bottom
  width=50% height=100%.
Primary Window
:p.
Here is the text for the primary window. This is a
:link reftype=hd res=004.
hypertext link
:elink.
to the secondary window.
:h2 res=004
  x=right y=bottom
  width=50% height=100%.
Secondary Window
:p.
Here is the text for the secondary window. This is a
:link reftype=hd res=003.
hypertext link
:elink.
```

to the primary window.

The origin of the first window is the lower left-hand corner of the coverage window. It occupies 50% of the width, but 100% of the height of the coverage window on the left-hand side.

The origin of the second window is the lower right-hand corner of the coverage window. It occupies 50% of the width, but 100% of the height of the coverage window on the right-hand side.

Although these two windows occupy adjacent positions on the screen, you cannot display them both at the same time. To define separate windows, you must specify a *group* number in the heading definition.

Displaying Multiple Windows

To display more than one window on the screen, you must assign a unique group number to each window with the **group=** attribute. This attribute can be specified with **:link.** or the heading tag.

If you do not specify a group number, a value of 0 is assigned. (This is the default value and is reserved for use by IPF.) If another window is already opened with the number specified for **group=**, IPF swaps its image (places the image in the same window) for the one defined by the heading or link tag.

Note: If a group number is assigned in both a heading and a hypertext or an automatic link, the link group number overrides the heading group number. The numbers you can assign to **group=** are integers from 1 to 64 000.

Compare the three heading definitions in the following figure. Notice that:

- The first and second windows have different group numbers and different positions.
- The second and third windows have the same group number.
- The second and third windows have the same size and position.

Source File for Window Group Number

```
:hl res=005
  x=left y=bottom
  width=50% height=100%
  group=1.
My First Window
:p.
Here is the text for the first window.
This is a
:link reftype=hd res=006.
hypertext link
:elink.
to the second window.
:p.
This is a
:link reftype=hd res=007.
hypertext link
:elink.
to the third window.
:hl res=006
  x=right y=top
  width=50% height=100%
  group=2.
My Second Window
:p.
Here is the text for the second window.
This is a
:link reftype=hd res=005.
hypertext link
:elink.
to the first window.
:p.
This is a
:link reftype=hd res=007.
hypertext link
:elink.
to the third window.
:hl res=007
  x=right y=top
  width=50% height=100%
  group=2.
```

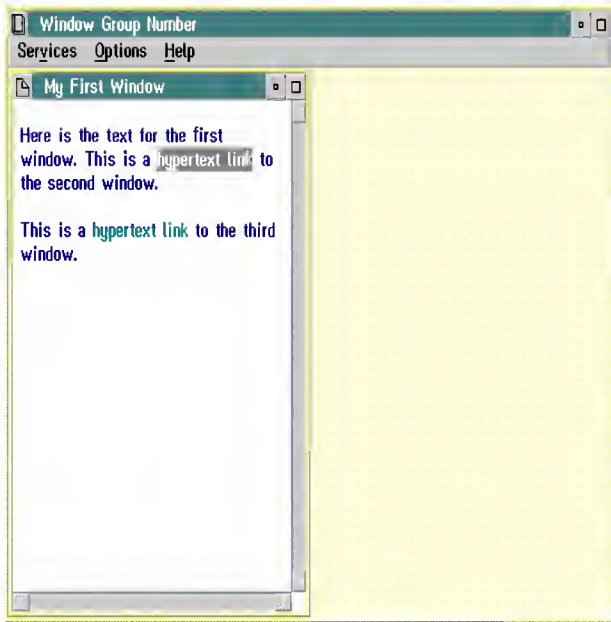


```

My Third Window
:p.
Here is the text for the third window.
This is a
:link reftype=hd res=005.
hypertext link
:elink.
to the first window.
:p.
This is a
:link reftype=hd res=006.
hypertext link
:elink.
to the second window.

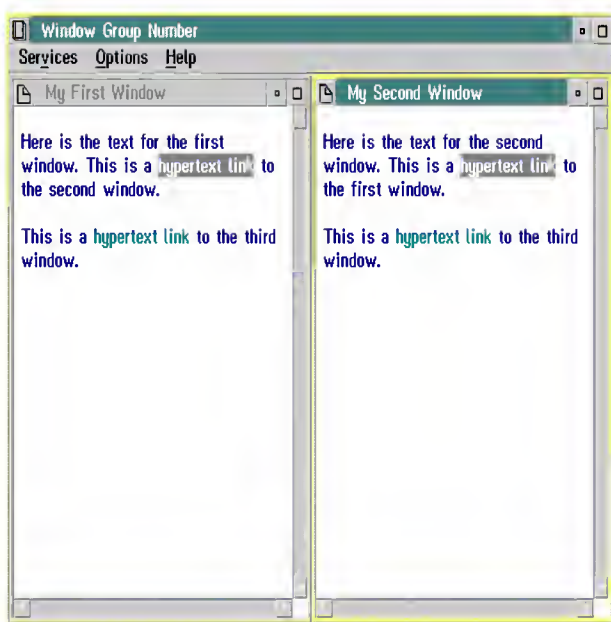
```

Now assume that the source file shown in the previous figure is compiled, and the user selects "My First Window" from the Contents window. The window in the following figure displays.



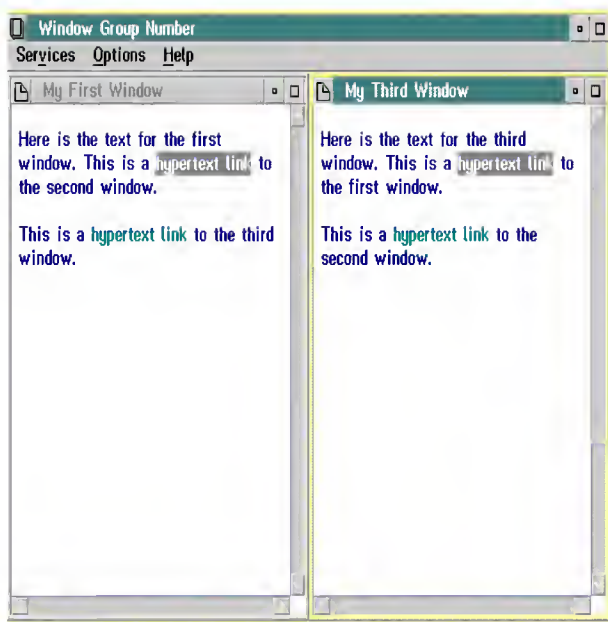
Multiple windows display with different group numbers.

If the user selects the hypertext link in this window, "My Second Window" will appear, as shown in the following figure.



Multiple windows display with same group numbers.

The windows appear next to each other because their heading definitions specify different group numbers. If the user now selects the hypertext link in "My Second Window," the resulting screen will be as shown in the following figure.



Compiled Output of Third Window from Group Number. "My Third Window" replaced "My Second Window" because it has the same group number as "My Second Window."

Preventing Image Swapping in Windows

The **group=** attribute opens a new window only if no other window with the same group number is already displayed. When a window is opened and a user selects another window with the same group number, IPF swaps its image in the already opened window. To prevent this, use the **viewport** attribute; it always opens a window.

Suppose you have defined the following hypertext link to a window:

```
:link reftype=hd res=001.  
    vpx=25% vpy=bottom  
    vpcx=75% vpcy=100%  
    viewport group=2.
```

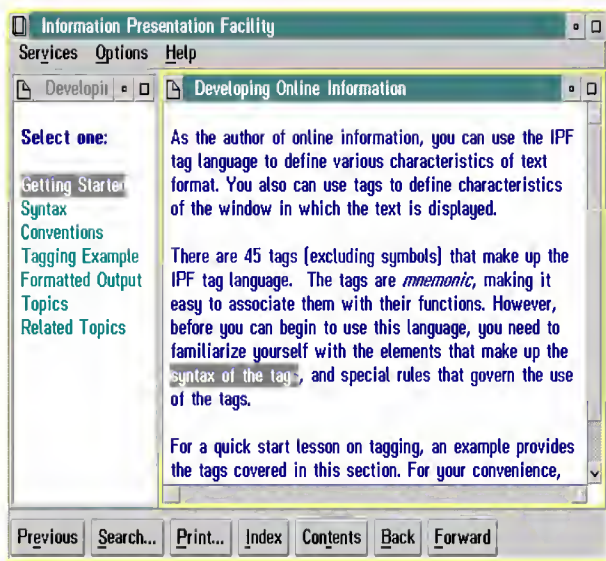
Guidance

```
:elink.
```

When this window is displayed, if the user selects the same hypertext link, the same window will open. You cannot control how many times the user will select a hypertext link. If you do not want another window opened each time the user selects the same hypertext link, use the **group=** attribute instead of the **viewport** attribute. This eliminates the potential for the user to open multiple windows containing the same information.

Linking to a Window Automatically

As we have seen, one way to display a secondary window is to enable the user to select a hypertext link from one window to another. Another way is to link the user to the secondary window automatically. For example, in the following figure the window on the right is displayed automatically when the user selects the window on the left (perhaps from the Contents window).



Example of a window displayed automatically. The window on the right was displayed automatically.

Auto Attribute

A window that starts the concurrent display of one or more other windows by automatic or hypertext links is referred to as the *owner* of the window chain. The **auto** attribute and the **reftype=hd** attribute indicate that a window is to be opened automatically whenever the owner window is opened. The **group=** attribute specifies the number of the window. (For more information about group numbers, see [Displaying Multiple Windows](#).)

The **vpx**, **vpy**, **vpcx**, and **vpcy** attributes indicate the size and position of the window in relation to its coverpage window.

Caution:

When defining automatic links, you do not want to create an "infinite loop" by linking to the same window or group number more than once in a chain of links.

For example, suppose you create three windows, A, B, and C, that contain the following automatic links.

Window A	Window B	Window C
> Link to B	> Link to C	> Link to A

When the file containing these links is compiled, the IPF compiler does not return an error message because of the loop. Now suppose **Window A** is an entry in the Contents window and the user selects it. Windows A, B, and C open and close uncontrollably until an error occurs and the process is terminated by the system.

Closing a Window Automatically

The **dependent** attribute causes the window to close automatically when the owner window is closed.

In the following example, the link at the end of the heading definition defines the owner window on the left. It links to the window on the right. Notice the link tag defining the automatic link does not require **:elink..**

```
:h1 res=421
    x=left y=bottom
    width=25% height=100%
```

```

        group=1.
Developing Online Information
:link reftype=hd res=422
    vpx=right vpy=bottom
    vpcx=75% vpcy=100%
    auto dependent group=2.
.
.
.
:h1 res=422.Developing Online Information

```

Tagging Example for Automatic Windows

The following tagging example defines two automatic window chains. A window chain has at least one owner window, and an owner window has one or more automatic or hypertext links to other windows in the chain. When an owner window closes, the windows in its chain that have specified the **dependent** attribute also close.

In "Example 1," the only owner window in the chain is the first window (**res=008**). It contains links to three other automatic windows, which are referred to as *sibling windows* of the owner window.

In "Example 2," Windows 1 through 3 in the chain are owner windows. Window 1 owns all the windows in the chain and can close all of them. Window 2 also owns Windows 3 and 4. Window 3 also owns Window 4, the last window in the chain, which is displayed by means of a hypertext link in the text.

```

:h1 Automatic Windows
:h2 res=008
    x=left y=top width=25% height=100%
    scroll=none group=1 clear.
Example 1
:link reftype=hd res=009
    vpx=25% vpy=top vpcx=25% vpcy=100%
    group=2 auto dependent.
:link reftype=hd res=010
    vpx=50% vpy=top vpcx=25% vpcy=100%
    group=3 auto dependent.
:link reftype=hd res=011
    vpx=75% vpy=top vpcx=25% vpcy=100%
    group=4 auto dependent.

:p.
This is Window 1.
:p.
This window has three automatic links to
Windows 2, 3, and 4.
:h2 res=009
    x=25% y=top width=25% height=100%
    scroll=none hide.
Window 2
:p.
This is Window 2.
:h2 res=010
    x=50% y=top width=25% height=100%
    scroll=none hide.
Window 3
:p.
This is Window 3.
:h2 res=011
    x=75% y=top width=25% height=100%
    scroll=none hide.
Window 4
:p.
This is Window 4.

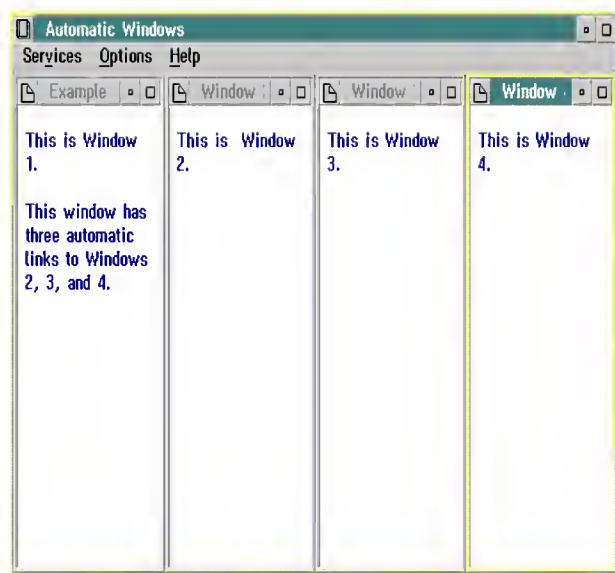
:h2 res=012
    x=left y=top width=25% height=100%
    scroll=none group=1 clear.
Example 2

```

```

:link reftype=hd res=013
  vpx=25% vpy=top vpcx=25% vpcy=100%
  group=2 auto dependent.
:p.
This is Window 1.
:p.
This window has an automatic link to
Window 2.
:h1 res=013
  x=25% y=top width=25% height=100%
  scroll=none hide.
Window 2
:link reftype=hd res=014
  vpx=50% vpy=top vpcx=25% vpcy=100%
  group=3 auto dependent.
:p.
This is Window 2.
:p.
This window has an automatic link to
Window 3.
:h1 res=014
  x=50% y=top width=25% height=100%
  scroll=none hide.
Window 3
:p.
This is Window 3.
:p.
This paragraph contains a
:link reftype=hd res=015
  vpx=75% vpy=top vpcx=25% vpcy=100%
  group=4 dependent.
hypertext link
:elink.
to Window 4.
:h1 res=015
  x=75% y=top width=25% height=100%
  scroll=none hide.
Window 4
:p.
This is Window 4.

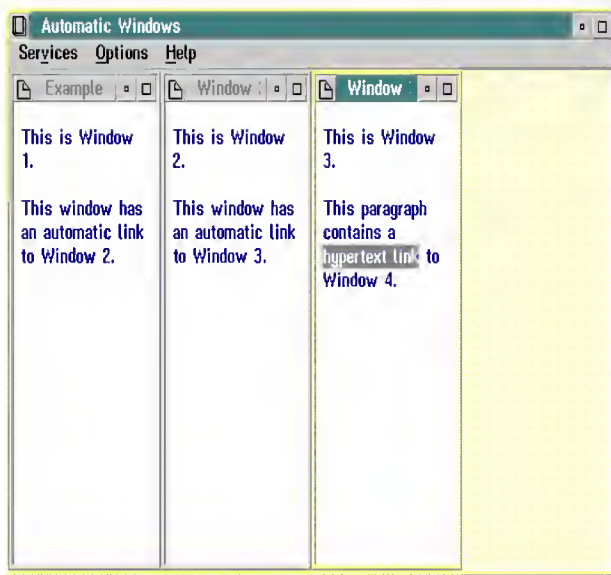
```



Example of four automatic windows.

When "Example 1" is selected from the Contents window, four windows are displayed in rapid succession. When Window 1 is closed, all four windows close.

The following figure shows the windows that are displayed when "Example 2" is selected from the Contents window.



Example of three automatic windows.

Notice Window 4 is not displayed. To display Window 4, you must select the hypertext link in Window 3.

Note: You can use the **viewport** attribute on an automatic link, because an automatic link is made only once.

Split Windows

A group of windows can be given the semblance of one window and yet offer the advantage of different windows; for example, text can be displayed next to an object the text describes. The author creates this effect by defining a window that consists of:

- **:h1.** or **:h2.** primary-window heading tags, followed by automatic links to secondary windows. (Text is not allowed.)
- **:h2.** secondary-window heading tags, each followed by text.

The primary window and its secondary windows must reside in the same file.

The position and size of the primary window determines the boundaries for its secondary windows. If the position and size of a secondary window are defined in absolute values that exceed the perimeter of the primary window, the secondary window is clipped. (When a window is clipped, part of it lies outside the window boundary and cannot be viewed.)

Sizes of secondary windows can be defined as percentages of the primary-window size. The minimum size of a secondary window (expressed in percentages) is zero height by zero width. Negative values for origin and position are not allowed.

Defining Split Windows

The primary window cannot have any text or graphics, only automatic links to each of its secondary windows. Each automatic link to a secondary window requires the **auto** and **split** attributes. The following is an example of the tagging for a primary window that contains a split window:

```
:h1 res=001 scroll=none.Primary Window A
:link reftype=hd res=002 auto split group=10
    vpx=left vpy=top vpcx=50% vpcy=100%
    scroll=none titlebar=none.
:link reftype=hd res=003 auto split group=11
    vpx=right vpy=top vpcx=50% vpcy=100%
    scroll=vertical titlebar=none.
```

The primary window does not have text and does not need a scroll bar; thus, the heading tag attribute is **scroll=none**. The primary window

can define an overall title bar and disable the title bars in the secondary windows.

Caution:

When defining split windows, do not link to a footnote from a secondary window.

For example, the text of a secondary window cannot have a link such as the following:

```
:link reftype=fn
      refid=001.
Display Pop-Up Window
:elink.
```

Tagging Example for Split Windows

The following tagging examples show the tagging for two different split-window arrangements.

In the first example, "Primary Window A" (**res=016**) has automatic links to two secondary windows, (**res=017** and **res=018**). The tagging for **res=017** has two list items, each of which is a hypertext link. The first list item, "Ducks," links to **res=018**; the second item, "World," links to **res=019**.

The tagging for both **res=018** and **res=019** refer to bit-map files.

Notice that in "Primary Window A" the link tags for the secondary windows specify **titlebar=none**, but the heading tags for the secondary windows specify "Dummy" as title text. You must always provide IPF with a title string in a heading tag, even when you specify that the window will not have a title bar and will not have an entry in the Contents window because you have specified the **hide** attribute. The link tags for a hypertext link to a secondary window must specify the **split** attribute. If the **split** attribute is omitted, the window will not behave as a secondary window; that is, it will not close when the primary window is closed, and instead of moving when the primary window is moved, it will become obscured.

Hide, Noprint, and Nosearch Attributes

In the examples, each secondary window heading has the **hide**, **noprint**, and **nosearch** attributes. The **hide** attribute prevents an entry from appearing in the Contents window. You do not want a secondary window (in a split-window arrangement) to be displayed by itself; you want it displayed only when the Contents entry for its primary window is selected.

The **nosearch** attribute prevents the title string of the secondary window from being listed as an entry in the Search Results window. The Search option of IPF also searches the secondary window (for a word or phrase) because of the link definition in the primary window; however, only the title string of the primary window is returned in the Search Results window.

The Print option of IPF enables the user to print one or more topics, the index, or the table of contents. The **noprint** attribute in a primary-window heading prevents the contents of a secondary window from being printed. Secondary windows are printed as part of their primary window. The contents of secondary windows are printed only in the order in which the link definitions appear in the primary-window definition.

None of the primary-window heading tags specifies a group number with the **group=** attribute, so IPF assigns 0 (the default) as the group number of each. The **clear** attribute causes the screen to be cleared of windows before each split window is displayed.

```
:h1 res=016 scroll=none clear.
Primary Window A
:link reftype=hd res=017 auto split group=10
      vpx=left vpy=top vpcx=50% vpcy=100%
      rules=border scroll=none titlebar=none.
:link reftype=hd res=018 auto split group=11
      vpx=right vpy=top vpcx=50% vpcy=100%
      rules=border scroll=none titlebar=none.
:h2 res=017 hide nosearch noprint.Dummy
:p.
This secondary window contains hypertext links
to the adjacent secondary window.
:p.
Select one:
```

```

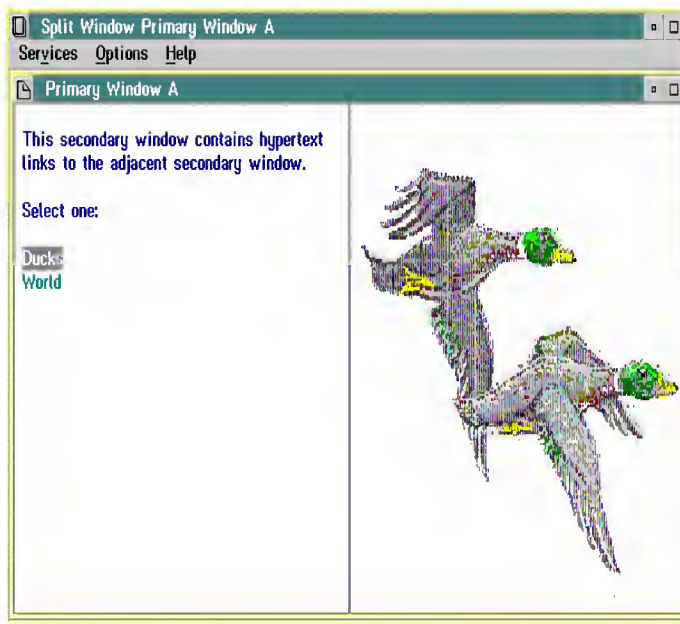
:sl compact.
:li.:link reftype=hd res=018 split group=11
      vpx=right vpy=top vpcx=50% vpcy=100%
      rules=border scroll=none titlebar=none.

Ducks
:elink.
:li.:link reftype=hd res=019 split group=11
      vpx=right vpy=top vpcx=50% vpcy=100%
      rules=border scroll=none titlebar=none.

World
:elink.
:esl.
:h2 res=018 hide nosearch noprint.Dummy
:artwork name='ducks.bmp' fit.
:h2 res=019 hide nosearch noprint.Dummy
:p.
:artwork name='world.bmp' fit.

```

Here are both views of the compiled version of Primary Window A.



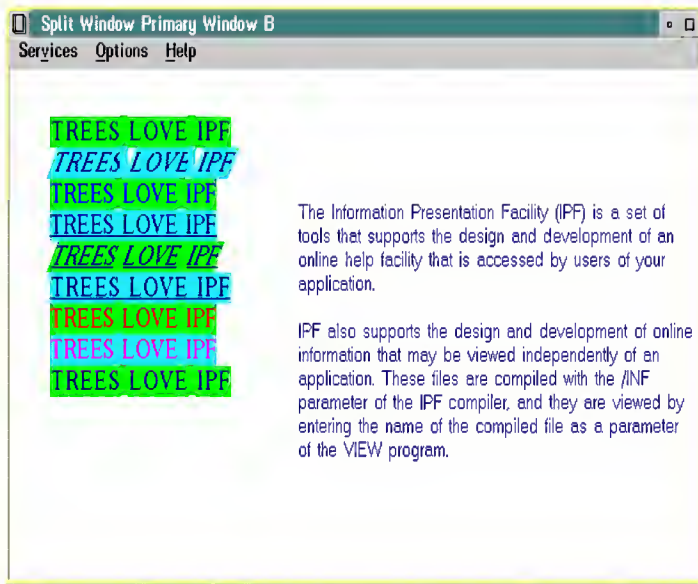
Example of a split window with an automatic link. The window on the right is displayed automatically when "Primary Window A" is selected from the Contents window.



Example of a split window with hypertext link. The window on the right is displayed when the second hypertext link is selected.

```
:h1 res=022 scroll=none titlebar=none rules=none clear.
Primary Window B
:link reftype=hd res=023 auto split group=10
    vpx=left vpy=top vpcx=40% vpcy=100%
    scroll=none titlebar=none rules=none.
:link reftype=hd res=024 auto split group=11
    vpx=right vpy=top vpcx=60% vpcy=20%
    scroll=none titlebar=none rules=none.
:link reftype=hd res=025 auto split group=12
    vpx=right vpy=bottom vpcx=60% vpcy=80%
    scroll=none titlebar=none rules=none.
:h2 res=023 hide nosearch noprint.Dummy
:lm margin=5.
:rm margin=2.
:p.
:font facename='Tms Rmn' size=24x12.
:color bc=green.:hp2.TREES LOVE IPF:ehp2.
:color bc=cyan.:hp3.TREES LOVE IPF:ehp3.
:color bc=green.:hp4.TREES LOVE IPF:ehp4.
:color bc=cyan.:hp5.TREES LOVE IPF:ehp5.
:color bc=green.:hp6.TREES LOVE IPF:ehp6.
:color bc=cyan.:hp7.TREES LOVE IPF:ehp7.
:color bc=green.:hp4.TREES LOVE IPF:ehp4.
:color bc=cyan.:hp3.TREES LOVE IPF:ehp3.
:color bc=green.:hp2.TREES LOVE IPF:ehp2.
:h2 res=024 hide nosearch noprint.Dummy
:p.
:h2 res=025 hide rules=none nosearch noprint.Dummy
:rm margin=3.
:font facename='Helv' size=18x9.
:p.
The Information Presentation Facility (IPF) is a set of tools
that supports the design and development of an online help
facility that is accessed by users of your application.
:p.
IPF also supports the design and development of online
information that may be viewed independently of an application.
These files are compiled with the /INF parameter of the IPF
compiler, and they are viewed by entering the name of
the compiled file as a parameter of the VIEW program.
```

Here is the compiled version of Primary Window B.



Example of a split window without window controls. You cannot see the boundaries of the three windows because the window controls were eliminated.

Push Buttons for Split Windows

Be careful when using heading tags to define a control area for split windows. A control area cannot be defined in the secondary window heading tag of a split window. You must define the control area (the coverage window) in the primary window heading tag. In the previous examples of split windows, the push button feature of IPF was disabled (see [Disabling the Display of Push Buttons](#)).

Summary Tables of Attribute Values for Origin and Size

The following tables summarize the attribute values that define a window's origin and size. Values shown in uppercase are keywords (words with special significance to IPF). Values shown in *lowercase italics* are to be substituted with your own values. Values are stacked when more than one value can be used with the attribute.

Attribute=Value	Description
X=CENTER LEFT RIGHT	Specifies the location of the x-axis. The x-axis runs horizontally from left to right.
Y=CENTER TOP BOTTOM	Specifies the location of the y-axis. The y-axis runs vertically from bottom to top.
WIDTH= <i>an integer followed by the percent sign (%)</i>	Specifies the width (horizontal space) of a window.
HEIGHT= <i>an integer followed by the percent sign (%)</i>	Specifies the height (vertical space) of a window.

Attribute=Value	Description
-----------------	-------------

VPX=CENTER LEFT RIGHT	Specifies the location of the x-axis. The x-axis runs horizontally from left to right. Overrides the x-axis attribute value specified by the heading tag.
VPY=CENTER TOP BOTTOM	Specifies the location of the y-axis. The y-axis runs vertically from bottom to top. Overrides the y-axis attribute value specified by the heading tag.
VPCX= <i>an integer followed by the percent sign (%)</i>	Specifies the width (horizontal space) of a window. Overrides the width attribute value specified by the heading tag.
VPCY= <i>an integer followed by the percent sign (%)</i>	Specifies the height (vertical space) of a window. Overrides the height attribute value specified by the heading tag.

Point to Remember Origin and size attributes in a link tag override the origin and size attributes in a heading tag.

Summary Table for Heading Attributes

This table summarizes the heading attributes that support a multiple-window format.

Attribute=Value	Function
res= id= name= global tutorial	Define references to internal and external sources.
x= y= width= height=	Define the origin and size of a window in relation to its coverage or primary window.
titlebar= scroll= rules=	Define the control the user has over the window.
viewport group= clear	Manage the display of information in multiple windows.
hide nosearch noprint	Restrict user retrieval of information.
toc=	Change heading levels appearing in the Contents window.

Summary Table for Link Attributes

This table summarizes the link attributes that support a multiple-window format.

Attribute=Value	Function
-----------------	----------

reftype= res= refid= database= object= data=	Define references to internal and external sources.
vpx= vpy= vpcx= vpcy=	Define the origin and size of a window in relation to its coverpage or primary window.
titlebar= scroll= rules=	Define the control the user has over the window.
viewport group= dependent auto split	Manage the display of information in multiple windows.

Points to Remember Link-tag attributes that have the same functions as those specified in a heading tag will override the heading-tag attributes. Although link-tag attributes have different names for x- and y-coordinates and window width and height, they provide the same functions.

Compiling Source Files

This chapter explains how to prepare your source files so that they will be recognized by the IPF compiler. This chapter also shows you how to enter the compile command, how to interpret error messages, and how to view the compiled document. A section on national language support (NLS) is also provided.

Source File Requirements

Using a single source file, you can produce a successful display of information with a limited number of tags. These tags are:

```
:userdoc.  
:docprof.  
:title.  
:h1.  
:p.  
:euserdoc.
```

The **:userdoc.** tag is always the first item in your source file. It identifies the beginning of an IPF file. This tag is a signal to the IPF compiler to begin translating the tag language.

The **:euserdoc.** tag is required as the last line of the file to signal the end of the tagged document.

Place the **:docprof.** tag at the beginning of your source file after the **:userdoc.** tag and before any heading definitions. Use the function of the **toc** (table of contents) attribute of the **:docprof.** tag to control the heading levels displayed in the Content window. For example, if you want only heading levels 1 and 2 to appear, the tagging is:

```
:docprof toc=12.
```

If no **toc=** value is specified, heading level 1 through 3 appear in the Contents window.

Not to be confused with window titles, the text string specified with a **:title.** tag is placed into the title bar of an online document. When the online document is displayed, the title appears on the title line of the main window. The tagging looks like this:

```
:title.Endangered Mammals
```

The maximum length of a title string specified with a **:title.** tag is 47 characters, including spaces and blanks.

The title tag provides a name for the online document but is also used for titles of Help windows. The title appears in the title bar of the main window. You usually place the title tag after the **:docprof.** tag.

Every file must start with a **:h1.** tag. Heading level sequences must not skip a level in the heading hierarchy. For example, you cannot have a heading level 1 tag (**:h1.**) followed by a heading level 3 tag (**:h3.**).

You must have at least one paragraph tag (**:p.**) and associated text to display a window.

The following figure shows an IPF source file.

The source file contains a **:userdoc.** tag, a **:title.** tag, a heading tag with a window identifier, a **:p.** tag, and the **:euserdoc.** tag.

```
. *
:userdoc.
:title.Endangered Mammals
:h1 res=001.The Manatee
. *
:p.
The Manatee has a broad flat tail and two flipper
like forelegs. There are no back legs.
The Manatee's large upper lip is split in two and
can be used like fingers to place food into the
mouth. Bristly hair protrudes from its lips,
and almost buried in its hide are small eyes, with
which it can barely see.
. *
:euserdoc.
```

Source File Structure

Source File Limits

- The maximum size of a line in an IPF source file is 255 characters.
- The maximum number of fonts in a source file is 16.
- The maximum number of unique words in a compiled file is 64 000.
- The maximum number of unique words in a panel is 64 000.
- The maximum number of panels is 64 000.
- The maximum number of external databases is 255.

These maximum limits are not absolute. Activity in the operating system can cause them to vary.

Naming Conventions

It is a good idea to give your source files an extension of IPF, so they can be distinguished from your other files. For example:

```
MYHELP.IPF
```

The IPF compiler does not require an IPF file-name extension; however, if your file has an IPF file-name extension, you will not have to type the extension at compile time.

Naming Restriction

During the compilation process, IPF creates some files to hold data temporarily, and erases the files when it no longer needs them. The names of these files are:

```
filename.clf
$0000$ (footnotes)
$1111$ (cross references)
$2222$ (tables)
$3333$ (bit maps)
```

where *filename* is the name of your source file. Do not give your source file a CLF extension. Also, do not give your source file a name of \$0000\$, \$1111\$, \$2222\$, or \$3333\$.

Do not use the name of any of the OS/2 environment variables for your file name. VIEW checks for an environment variable that matches its first argument and, if it finds one, such as CPREF, it will take the value of that environment variable and use the value as the name of the file or files to open.

Using a Base Source File

The IPF compiler can produce a single output document by processing multiple input files through one base source file. This process is most often associated with online documents. For example, the online *Information Presentation Facility Reference* has more than ten separate source files, but all the files were processed through one base file.

The **.im** (imbed) control word sends a signal to the compiler and tells it to process each file in the sequence listed in the base file.

A portion of the base file IPFCBASE.IPF for the online *Information Presentation Facility Reference* looks like this:

```
:userdoc.
.
.
.
.im ipfcch01.ipf
.im ipfcch02.ipf
.im ipfcch03.ipf
.
.
.
```

The placement of an imbedded file determines the order of entries in the table of contents.

Imbedded files cannot use the **:userdoc.** or **:euserdoc.** tags.

Note: When using a base source file to process multiple files, enter the base file name as the *filename* parameter of the IPFC command.

Starting the IPF Compiler

You can start the IPF compiler and specify all input from the command line. A new command-line interface has been introduced. To view it,

type **IPFC** with no parameters.
The syntax is:

Usage:

```
IPFC [-switch]... [-option]... filename [outfile]
```

The switches perform the functions in the following list:

Switches:

- i - Compiles the source file as an online document
- s - Suppresses the performance of the search function
- x - Generates and displays a cross-reference list

The options perform the functions in the following list:

Options:

- W:level - Warning level
- D:dbcscode - Country code
- C:codepage - Character code page
- L:language - Language ID

The following parameters provide international language support:

-d: *nnn* The *nnn* is the 3-digit country code.

-c: *nnnn* The *nnnn* is the 4-digit code page.

-l: *xxx* The *xxx* is a 3-letter identifier that indicates the language file to be used.

See [Country Code, Code Pages, and Language Parameters](#) for the tables with the NLS values.

For example:

```
IPFC myfile.txt /INF -d:033 -c:0437 -l:FRA
```

If you do not specify these parameters, the default for -d: nnn and -c: nnnn are the values specified in your CONFIG.SYS file.

For help on any of the options, type

```
IPFC -X:?
```

where "X" is one of the options.

The outfile parameter is used to specify the name of the output file. If this parameter is not used, the output file will have the same filename as the input file and an extension of either INF or HLP. For example:

```
IPFC TEST.IPF C:\FILE.HLP
```

Command Line Interfaces For Previous Versions of the IPF Compiler

The interface from earlier levels of the compiler is still supported. The syntax is:

```
IPFC filename [/INF] [/S] [/X] [/W] [> messageoutputfilename] [/COUNTRY] [/CODEPAGE] [/LANGUAGE]
```

where:

filename Specifies the name of your IPF source file or base file.

If you do not give a file-name extension, the IPF compiler uses .IPF by default. If your file has a file-name extension other than IPF, include that file-name extension in the command line.

/INF Compiles the source file as an online document.

If this parameter is not included, the default is to compile the source file as a help library, whose extension is .HLP.

/S	Suppresses the performance of the Search function. This parameter increases compression of compiled data by about 10% to further reduce the storage it requires.
/X	Generates and displays a cross-reference list.
/Wn	Generates and displays a list of error messages. The <i>n</i> indicates the level of error messages you want to receive. Values you can specify for <i>n</i> are 1, 2, or 3. The default is W3. For more information, see Interpreting IPFC Error Messages .
messageoutputfilename	Specifies the name of the file where error and cross reference messages are sent. If you do not specify this parameter, messages generated by /X and /Wn are sent to the display screen.

The following parameters provide international language support (NLS):

/COUNTRY=nnn	The <i>nnn</i> is the 3-digit country code.
/CODEPAGE=nnnn	The <i>nnnn</i> is the 4-digit code page.
/LANGUAGE=xxx	The <i>xxx</i> is a 3-letter identifier that indicates the language file to be used.

See [Country Code, Code Pages, and Language Parameters](#) for the tables with the NLS values.

For example:

```
IPFC myfile.txt /INF /COUNTRY=033 /CODEPAGE=437 /LANGUAGE=FRA
```

If you do not specify these parameters, the default for /COUNTRY and /CODEPAGE are the values specified in your CONFIG.SYS file.

Compiling Help Files

To compile a source file that is intended as a help-text window, use the IPFC command without the `-i` switch or `/INF` option. For example:

```
IPFC myhelp.txt
```

Environment Variables

There are four environment variables that can be used to specify the location of source files. The IPFC environment variable is used to specify where the IPF support files (such as the *.NLS files) are stored. The IPFCIMBED environment variable is used to search for files imbedded with the .im macro. The IPFCARTWORK environment variable is used to specify the location of artwork files and artlink files. The TMP environment variable is used to indicate where the compiler should store the intermediate files it creates during the compilation.

The IPFC environment variable has been enhanced to allow multiple paths.

Note: IPFCARTWORK=artwork and linkfile path (used for :artwork.)
 IPFCIMBED=imbed file path (used for .im)
 IPFC=IPFC file path (location of APSYxxxx.APS, IPF*.NLS)
 TMP=temporary file path

Viewing an Online Document

If you want to see your formatted online document, you can use the VIEW command to display it.

An online document has an extension of INF. It can be viewed by entering its name as a parameter to the VIEW command; for example:

```
VIEW myfile
```

You do not need to include the INF file extension.

You cannot use VIEW to display help-text windows for application programs.

Where IPFC Files are Stored

When you first install the Toolkit, the following environment variable is placed into the CONFIG.SYS file:

```
IPFC=C:\TOOLKIT\IPFC
```

The IPFC environment variable identifies the directory in which data files needed by the IPF compiler are stored.

When you first install the system, the following environment variables are placed into the CONFIG.SYS file:

The HELP environment variable identifies the location of .HLP libraries.

```
Help=C:\OS2\HELP
```

The BOOKSHELF environment variable identifies the location of online documents and is used by VIEW.

```
BOOKSHELF=C:\OS2\BOOK
```

Concatenating .INF Files

Concatenation of .INF files is useful when you have a large amount of information that cannot be compiled as one file that fits on a diskette. If you want to concatenate files, you must use the **res=** attribute for window identifiers.

After you have created your .INF files, use the SET command to set an environment variable equal to a string that consists of the .INF file names, for example:

```
SET PROGREF=PRINTRO.INF+PRCP.INF+PRWIN.INF+PRDATA.INF
```

When you specify the environment value as a parameter to the VIEW program, VIEW displays the online information. Headings from the different files are displayed in the contents window in the order the files are concatenated for the environment variable.

Interpreting IPFC Error Messages

The **-W:n** or **/Wn** option of the IPFC command determines the levels of error messages that will be displayed. Following are the values that you can specify for *n*:

Value	Meaning
1	Returns only warning level 1 messages. Warning level 1 messages are the most severe.
2	Returns warning level 1 and 2 messages.
3	Returns all three warning levels of messages. This is the default. Warning level 3 messages are the least severe.

When IPF compiles your file, it generates and displays the error messages. If no errors are found, IPF tells you that compiling has been completed and no errors were found.

You may prefer to redirect error messages from the screen to an error file. You could enter the IPFC command like this:

```
IPFC myhelp -w:3 > myhelp.err
```

If you have also requested that a cross-reference list be created by specifying the `-X` switch, it will be included in the MYHELP.ERR file.

For a list of error messages that the IPF compiler returns, see [Compiler Error Messages](#).

Differences between .HLP and .INF Files

	Help Libraries	Online Documents
IPFC Command Syntax	IPFC filename	IPFC filename.INF -i
Compiled File Extensions	.HLP	.INF
Environment Variables Used by the VIEW Program		BOOKSHELF= defines the location of .INF files
Environment Variables Used by Help Manager for Help Windows	HELP= defines the location of .HLP libraries.	
Cause of Interface Display	An application user's request for help.	Entering the file name as a parameter to the VIEW utility.
Initial Size of Main Window	35% of screen (default)	85% of screen (default)
Initial Contents of Main Window	Response to help request	Contents window (default)
Main Window Title	Defined by the programmer in the HELPINIT structure.	Defined by the :title. tag, which is placed on the line following the :userdoc. tag.
External Links (see global attribute of heading tag and database attribute of :link. tag.)	.HLP files can link to other .HLP files and to .INF files by launching VIEW.	.INF files can link only to other .INF files.
To View Concatenated Files	Specify a string of .HLP files in the HELPINIT structure or send the HM_SET_HELP_LIBRARY_NAMES message.	Set an environment variable equal to a string of .INF file names.

Note: Defaults may be overridden by objects that are displayed in application-controlled windows.

National Language Support (NLS)

The following parameters provide national language support (NLS). These parameters were updated between Version 2.1 and 3.0 of OS/2. All of the parameters shown in the following table are supported in the current IPF compiler.

up to Version 2.1	Version 3.0 and higher
/COUNTRY = nnn	-d: nnn (nnn = a numeric value)
/CODEPAGE = nnnn	-c: nnnn (nnnn = a numeric value)
/LANGUAGE = xxx	-l: xxx (xxx = alphabetic letters)

If you do not specify these parameters, the default for /COUNTRY or -d: nnn and /CODEPAGE or -c: nnnn are the values specified in your CONFIG.SYS file.

Refer to the "Information Presentation Facility" chapter in the [OS/2 Bidirectional Language Support Developer's Guide and Reference](#).

Country Code, Code Pages, and Language Parameters

The following table lists the 3-digit country code for the /COUNTRY or -d: nnn parameter, the numeric identifiers of code pages, and the APS filename of the IPFC command supported.

Country	Country Code	Code Pages	APS File
Arabic	785	0864	APSY0864.APS
Australia	061	0437, 0850	APSYMBOL.APS
Belgium	032	0437, 0850	APSYMBOL.APS
Brazil	055	0850, 0437	APSYMBOL.APS
Bulgaria	359	0915	APSY0915.APS
Canadian English	001	0437, 0850	APSYMBOL.APS
Canadian French	002	0863, 0850	APSYMBOL.APS
Catalan	034	0850, 0437	APSYMBOL.APS
Chinese (Simplified)	086	1381	APSY1381.APS
Chinese (Simplified)	086	1386	APSY1386.APS
Chinese (Traditional)	088	0950	APSY0950.APS
Czech	421	0852	APSY0852.APS
Denmark	045	0865, 0850	APSYMBOL.APS
Finland	358	0437, 0850	APSYMBOL.APS
France	033	0437, 0850	APSYMBOL.APS
Germany	049	0437, 0850	APSYMBOL.APS
Greece	030	0869	APSY0869.APS
Greece	030	0813	APSY0813.APS
Hebrew	972	0862	APSY0862.APS
Hungary	036	0852	APSY0852.APS
Italy	039	0437, 0850	APSYMBOL.APS
Japan	081	0932, 0437, 0850	APSY0932.APS
Korea	082	0949, 0934	APSY0949.APS

Latin America	003	0437, 0850	APSYMBOL.APS
Lithuania	370	0921	APSY0921.APS
Netherlands	031	0437, 0850	APSYMBOL.APS
Norway	047	0865, 0850	APSYMBOL.APS
Poland	048	0852	APSY0852.APS
Portugal	351	0860, 0850	APSYMBOL.APS
Russia	007	0866	APSY0866.APS
Slovenia	386	0852	APSY0852.APS
Spain	034	0437, 0850	APSYMBOL.APS
Sweden	046	0437, 0850	APSYMBOL.APS
Switzerland	041	0437, 0850	APSYMBOL.APS
Thailand	066	0874	APSY0874.APS
Turkey	090	0857	APSY0857.APS
United Kingdom	044	0437, 0850	APSYMBOL.APS
United States	001	0437, 0850	APSYMBOL.APS

The following table lists the 3-letter identifier for the /LANGUAGE and -l: xxx parameter of the IPFC command:

ID	Language	NLS File
ARA	Arabic	IPFARA.NLS
BGR	Bulgarian	IPFBGR.NLS
CAT	Catalan	IPFCAT.NLS
CHT	Chinese (Traditional)	IPFCHT.NLS
CZE	Czech	IPFCZE.NLS
DAN	Danish	IPFDAN.NLS
DEU	German	IPFDEU.NLS
ELL	Greek 813	IPFELL.NLS
ENG	English UP	IPFENG.NLS
ENU	English US	IPFENU.NLS
ESP	Spanish	IPFESP.NLS
FIN	Finnish	IPFFIN.NLS
FRA	French	IPFFRA.NLS
FRC	Canadian French	IPFFRC.NL
GRK	Greek 869	IPFGRK.NLS
HEB	Hebrew	IPFHEB.NLS
HUN	Hungarian	IPFHUN.NLS
ITA	Italian	IPFITA.NLS
JPN	Japanese	IPFJPN.NLS
KOR	Korean	IPFKOR.NLS
LTU	Lithuanian	IPFLTU.NLS
NLD	Dutch	IPFNLD.NLS
NOR	Norwegian	IPFNOR.NLS

POL	Polish	IPFPOL.NLS
PRC	Chinese (Simplified) 1381	IPFPRC.NLS
PRC	Chinese (Simplified) 1386	IPFGBK.NLS
PTB	Brazilian/Portuguese	IPFPTB.NLS
PTG	Portuguese	IPFPTG.NLS
RUS	Russian	IPFRUS.NLS
SLO	Slovene	IPFSLO.NLS
SVE	Swedish	IPFSVE.NLS
THI	Thai	IPFTHI.NLS
TRK	Turkish	IPFTRK.NLS
UND	User defined	IPFUND.NLS

Note: If there is an APSYxxxx.APS file that matches the code page you are using to compile your IPF file (either specified or default), the IPFC will use that file. Otherwise, it will use APSYMBOL.APS file that is suitable for code page 437 or 850.

Creating New Code Pages

The following information tells you how to create an APSYxxxx.APS file and add it to the IPFC directory which allows you to add support for new code pages that you create.

Creating the Contents of a Code Page File

You can specify a symbol and associate a character with that symbol in a code page file as follows:

1. In the first column, place a character that you want to substitute with the symbol; for example, +.
2. Define a symbol with text placed between a character ampersand (&) and a period (.); for example, &plus..

Following are some examples for defining symbols:

```
*&asterisk.
.&period.
+&plus.
,&comma.
-&dash.
```

Each line in a file must not contain more than one symbol - enter the next symbol on a new line.

The symbol character must be in the first column of your file and its associated symbol must start in the second column.

The text in the symbols you define are case sensitive; for example, :&colon. and :&Colon. are two different symbols. The symbol defined for :&colon. cannot be used as :&Colon.. If you attempt to use them as two symbols, the compiler generates a warning message that a symbol is not found.

Naming the Code Page File

You name a new code page file APSYxxxx.APS where xxxx is a unique four-digit number. You can also use a four-digit number from the "Code Pages" column in the table "NLS Country, Code Page, and Language Parameters for IPF Compiler" for the country you want to use.

You substitute your new code page for that code page. For example, to create a new code page for a language called Pidgeon, an appropriate APS filename could be APSY0222.APS or APSYMBOL.APS.

Ensure that your APSYxxxx.APS file is located in the following directory:

\TOOLKIT\IPFC

Creating Files for Adding New Languages

You create an IPFxxx.NLS file when you want to add a new language to the compiler.

Note: Not all languages are supported - only use those that are similar to those already supported by IPFC.

Creating the Contents of a Language File

You can specify the following attributes in an IPFxxx.NLS file as follows:

Attribute	Description
REM	<p>Use this attribute to specify comments in an IPFxxx.NLS file. To enter multiple comments, specify REM on each new line followed by the text of the comment. For example:</p> <pre>REM This is a comment. REM This is another comment.</pre>
NOTE	<p>Use this attribute to translate the default text of the :note. tag. The text specified between the quotation marks is used instead of the default word Note. For example, to replace the default of Note: with Nota:, enter the following:</p> <pre>NOTE="Nota: "</pre>
CAUTION	<p>Use this attribute to translate the default text of the :caution. tag. The text specified between the quotation marks is used instead of the default word CAUTION. For example, to replace the default of CAUTION: with ATTENZIONE:, enter the following:</p> <pre>CAUTION="ATTENZIONE: "</pre>
WARNING	<p>Use this attribute to translate the default text of the :warning. tag. The text specified between the quotation marks is used instead of the default word Warning. For example, to replace the default of Warning: with Avvertenza:, enter the following:</p> <pre>WARNING="Avvertenza: "</pre>
DANGER	<p>Use this attribute to translate the default text of the :danger. tag. The text specified between the quotation marks is used instead of the default word Danger. For example, to replace the default of Danger: with Unsafe:, enter the following:</p> <pre>DANGER="Unsafe: "</pre>
OLCHARS="single character or a string"	<p>Use this attribute to specify the ordered list characters you want to use for secondary list entries. For example, to use the US alphabetic characters, specify the following:</p> <pre>OLCHARS="abcdefghijklmnopqrstuvwxyz "</pre>

The result is:

- 1. Entry one.
 - a. Line 1 of the secondary entry
 - b. Line 2 of the secondary entry
 - .
 - .
 - .

OLCHARS="abcdefghijklmnopqrstuvwxyz" is the default value.

OLCLOSE1="single character or a string"

Use this attribute to specify the character displayed after the number associated with each list item. For example, to use the character ")" after a number in the list entry, specify the following:

OLCLOSE1=")" "

The result is:

- 1) Entry one.
 - a. Line 1 of the secondary entry
 - b. Line 2 of the secondary entry
- 2) Entry two.
 - .
 - .
 - .

OLCLOSE1=. is the default value.

OLCLOSE2="single character or a string"

Use this attribute to specify the character displayed after the value associated with each secondary list item. For example, to use the character "-" after a number in the secondary list entry, specify the following:

OLCLOSE2="-" "

The result is:

- 1) Entry one.
 - a- Line 1 of the secondary entry
 - b- Line 2 of the secondary entry
- 2) Entry two.
 - .
 - .
 - .

OLCLOSE2=. is the default value.

ULITEMID1="single character or a string"

Use this attribute to specify the first level entries of an unordered list. For example, to use the character "-" for first level entries, specify the following:

ULITEMID1="-" "

The result is:

- Line 1 of the first level
- Line 2 of the first level
- .
- .
- .

ULITEMID1=" (Alt 249 in code page 437) is the default value.

ULITEMID2="single character or a string"

Use this attribute to specify the secondary level entries of an unordered list. For example, to use the character "+" for secondary level entries, specify the following:

ULITEMID2="+" "

The result is:

- Line 1 of the first level

```

      + Line 1 of the second level
      + Line 2 of the second level
- Line 2 of the first level
.
.
.

```

ULITEMID2=- is the default value.

ULITEMID3="single character or a string"

Use this attribute to specify the third level entries of an unordered list. For example, to use the character "oo" for third level entries, specify the following:

```
ULITEMID3="oo"
```

The result is:

```

- Line 1 of the first level
  + Line 1 of the second level
    oo Line 1 of third level
    oo Line 2 of third level
.
.
.

```

ULITEMID3=o is the default value.

GRAMMAR

Use this attribute to indicate the beginning of the grammar rules.

WORDS

Use this attribute to specify the set of characters that can make up a word. For example, to specify that a word can be made of alphanumeric characters, specify:

```
WORDS=0 - 9 + A - Z + a - z
```

WORDS=0 - 9 + A - Z + a - z is the default value for this attribute.

GRAPHCS

Use this attribute to specify the range of graphics characters that make up a graphic word.

GRAPHICS="\x01 - \x1F + \x80 - \xFF" is the default value for this attribute.

EGRAMMAR

Use this attribute to indicate the end of the grammar rules.

Naming the Language File

You name a new language file IPFxxx.NLS where xxx is unique for the language you want to use. Name your new file with xxx as different letters than any of the three-letters used in the "ID" column in the table "The 3-Letter Identifier for the /LANGUAGE and -l: xxx Parameter". For example, to create a new language called Pidgeon, an appropriate NLS filename could be IPFPID.NLS.

Ensure that your IPFxxx.NLS file is located in the following directory:

```
\TOOLKIT\IPFC
```

IPF Tag Reference

This section contains an alphabetic listing of the tags used by the IPF compiler to create online documents and help windows. An IPF tag controls the format of the displayed output.

The syntax description of each tag includes the tag name, the element that the tag describes, the attributes of the tag, and the end tag. A tag begins with a colon (:) and ends with a period (.). Most tags have an end tag associated with them. An end tag has the same name as the tag, preceded by the letter "e." For example, the end tag for the :**userdoc**. tag is the :**euserdoc**. tag.

A tag can have one or more attributes associated with it. An attribute provides additional control information for the tag. Some attributes are followed by an equal sign and a value. If the value contains blanks or special characters, it must be enclosed in apostrophes or single quotation marks. For example:

```
:font facename='Tms Rmn'.
```

Notice that the period that ends the tag follows the attributes specified for the tag. If no attributes are specified, the period immediately follows the tag name. For example, when the **:note.** tag does not have the **text=' '** attribute specified, the period immediately follows the word **:note.**

Some tags are required to be in a specific order before the file can be compiled by the IPF compiler. The following example shows the minimum tags required to compile a file:

```
:userdoc.  
:h1 id=example1.Tag Example 1  
:p.This is the first tag example.  
:euserdoc.
```

This section also describes control words used by the IPF compiler. Control words start with a period (.). A control word tells the IPF compiler about the statement that it is part of. For example, the imbed (.im) control word tells the IPF Compiler to include the specified file in the source file at this point.

.br (Break)

Purpose

Causes a break in a line of text.

Syntax

Control Word	Element	Attributes	End
.br	Break		

Attributes

None

Description

Use the **.br** control word to stop the display of text on a line, and continue it on the next line. The break control word must be the only statement on the line. If you enter text on the same line as the break control word, the IPF compiler ignores the break control word.

The break control word is especially useful before a line of text that contains a symbol.

Conditions

The **.br** control word must start in column 1, and be the only statement on the line.

Example

```
:p.These words  
appear on  
the same line.  
.br  
These words  
.br  
do not.
```

Output

These words appear on the same line.
These words
do not.

For more information, see [Break](#).

.ce (Center Text)

Purpose

Centers text.

Syntax

Control Word	Element	Attributes	End
.ce	Text		

Attributes

None

Description

The **.ce** control word allows you to center text on a line.

Conditions

The text cannot contain any IPF tags. It can, however, contain nameit symbols and IPFC symbols.

The **.ce** control word's centering effect is limited to text on the same line with the control word.

The line will remain centered as you resize the window unless the window becomes so small that the line must be broken, at which point centering justification is lost (when you resize the window to be large enough to hold the line again, centering is restored).

The control word must start in column 1.

Example

```
.ce The centered text must not contain any IPF tags.
```

Output

```
The centered text must not contain any IPF tags.
```

. * (Comment)

Purpose

Places a comment into a file.

Syntax

Control Word	Element	Attributes	End
.*	Comment		

Attributes

None

Description

The .* control word allows you to place a comment line into your file. The IPF compiler ignores any text on the same line as the comment control word, and does not display this text.

The comment control word must be the first statement on the line of text that you do not want displayed. Each comment control word must begin on a new line.

You can use comment control words to refer to items, to place notes into your file, or to prevent the display of an item.

No space is required between the comment control word and the text that follows it. Comment control words are used independently of IPF tags. They are not used between any IPF tags or with any IPF tag and its accompanying text or attributes.

Conditions

Do not use the comment control word:

- Within the IPF tag, that is, between the colon that starts the tag and the period that ends the tag.
- Between an IPF tag and its accompanying text or attributes.

Always start the comment control word in column 1.

Example

```
.* The comment control word must be the first statement on the line.  
.* When the source file is compiled, the text on the  
.* comment line is not displayed.
```

Output

When the file is compiled, the comment control word and the information following it on the comment line are not displayed.

.im (Imbed)

Purpose

Specifies that text files are to be included at process time.

Syntax

Control Word	Element	Attributes	End
.im	Imbed		

Attributes

None

Description

The **.im** control word enables you to include text files when you are ready to compile your file.

Conditions

- If the file to be included is not in the current directory, you must enter a complete file name.
- Imbedded files must not use the **:userdoc.** or **:euserdoc.** tag.

Always start the **.im** control word in column 1.

Example

```
:userdoc.  
.im filename.ext  
.im c:\main\filename.ext  
:euserdoc.
```

Output

The text and art in the imbedded files are displayed when you access the compiled file.

.nameit (Nameit)

Purpose

Defines symbols in a document.

Syntax

Control Word	Element	Attributes	End
.nameit	Nameit	symbol= text='string'	

Attributes

symbol=name

Identifies a name for the symbol you want to create. It can be up to 10 characters long (A-Z, 0-9), with no blanks or special characters; the first character must be a letter. The & character cannot be used in the name.

text='string'

Identifies the content of the value to be assigned to the symbol and is what is displayed. In addition to text, the text string can contain other nameit symbols, IPFC symbols, and IPFC tags.

Description

The **.nameit** control word enables you to create symbols.

Conditions

Do not use the & character as part of the name of the symbol. You must use the & character when you actually use the symbol.

Always start the nameit control word in column 1.

Example

```
.nameit symbol=os text='operating system'
.
:p.The &os. supports multitasking.
```

The output looks like this:

The operating system supports multitasking.

:acviewport. (Application-Controlled Window)

Purpose

Enables an application to dynamically control what is displayed in an IPF window.

Syntax

Tag	Element	Attributes	End
:acviewport.	Have IPF call a function in a dynamic-link module.	dll=' ' objectname=' ' objectinfo=' ' objectid=' '	
	Define the window in which the function runs.	vpx= vpy= vpcx= vpcy=	

Attributes

dll=' '
Specifies a dynamic-link module for IPF to load so that a *communication object* (a function) in the module can be run in a window (an *application-controlled window*).

objectname=' '
Identifies the entry point of the communication object in the dynamic-link module. The value specified for this attribute is case sensitive.

objectinfo=' '
Identifies parameters to be passed to the object.

objectid=' '
Specifies an identifier that will associate the window with the object. The application or communication object can get this value from the Object ID field in the ACVP data structure and use it to determine the panel that activated the viewport.

vpx=
vpy=
vpcx=
vpcy=

Define the location and size of the window. **vpx=** and **vpy=** are positions along the x (horizontal) and y (vertical) axes. The point where the values intersect represents the origin of the window. **vpcx=** and **vpcy=** represent changes along the x and y axes with respect to

the origin.

These attributes can be expressed as absolute values, relative values, or dynamic values:

Absolute value:

A number followed by a letter, which indicates the unit of measure:

c	(Characters): Average character width of the default system font.
x	(Pixels): Dependent on the display adapter in use.
p	(Points): Typesetting measure; equal to approximately 1/72 inch.

Relative value:

A number followed by the percent sign (%), indicating a percentage of the parent-window width or height.

Dynamic value:

A term indicating a window coordinate location that is dependent on the current size and position of the parent window:

left center right	For x values, flush left with, in the center of, or flush right with the parent window.
top center bottom	For y values, at the top, center, or bottom of the parent window.

Description

:acviewport. is used in either a help file or an online document file to specify that a window will be under the control of a routine that was written and compiled as part of a dynamic-link module. When an IPF window is selected for display at run time, and **:acviewport.** is encountered, IPF passes control to the entry point (**objectname=**) in the dynamic-link module. At this point, the routine in the module takes control. For more information, see [Customizing IPF with Communication Objects](#).

The definition for **:acviewport.** must follow a primary heading; for example:

```
:h2 res=2000
  x=left y=top width=100% height=100%
  scroll=none titlebar=both clear group=1.Information Windows
:acviewport dll='My_DLL' objectname='My_Routine' objectid='1'
  vpx=right vpy=top vpcx=50% vpcy=100%
```

In the example, a window is displayed within the primary window indicated by the heading tag (**:h2.**) and its attributes. The contents of the window are controlled by the communication object, **My_Routine**, in the dynamic-link module, **My_DLL**.

When the user selects the primary window and **:acviewport.** tag is encountered, IPF calls the communication object in the dynamic-link module and sizes the child window.

:artlink. (Art Link)

Purpose

Identifies link definitions for hypergraphic areas of a bit map, metafiles are not supported.

Syntax

Tag	Element	Attributes	End
:artlink.			:eartlink.

Attributes

None

Description

Use **:artlink.** in conjunction with the artwork tag (**:artwork.**) to indicate links to a bit map or segments of a bit map. The link definitions are specified by link tags (**:link.**) and follow **:artlink.**, as in Example 1.

Example 1

```
:artlink..  
:link reftype=hd res=001 x=0 y=0 cx=16 cy=8.  
:link reftype=fn refid=afnr x=16 y=8 cx=16 cy=8.  
:link reftype=inform res=0345 x=0 y=8 cx=16 cy=8.  
:eartlink.
```

(For more information, see [:link. \(Link.\)](#))

All of the above could be in a separate file, which would be identified by the **linkfile.** attribute of the artwork tag, as in Example 2.

Example 2

```
:artwork name='mybitmap.bmp' linkfile='mylinks'.
```

In this example, MYBITMAP.BMP is the name of the file containing the bit map, and MYLINKS is the file consisting of the entries shown in Example 1.

If the artwork tag does not specify the attribute **linkfile=**, IPF looks for **:artlink.** on the line immediately following **:artwork.**, as shown in Example 3.

Example 3

```
:artwork name='mybitmap.bmp'.  
:artlink..  
:link reftype=hd res=001.  
:eartlink.
```

In this example, if the user clicks on the bit map associated with this art link, the window with the identifier, 001 is displayed.

If no **:artlink.** tag is found, no hypergraphic areas for the bit map are defined.

You can divide a bit map into rectangular segments, each of which is selectable and links to different information. For each segment, you need to define values for x, y, cx, and cy, which represent pixel values on the x and y axes. The x axis is always horizontal, and the y axis is always vertical; x and y define the origin of the segment, while cx and cy identify the changes in x and y. The value 0,0 indicates the origin of the bit map and is always the bottom-left corner.

Following is an example of a segmented bit map.

```
0,16                                     32,16  
|  
|  
|  
y  
|  
|  
|  
0,0 -----x----- 32,0
```

Example 4 shows the tagging when the link is from a segmented bit map. The name of the segmented bit-map file is **show2.bmp**; the name of the file with the link information is **link.dat**.

Example 4

```
:artwork name='show2.bmp' linkfile='link.dat'.
```

The following information could be placed into LINK.DAT.

```
:artlink..  
:link reftype=hd res=001 x=0 y=0 cx=16 cy=8.  
:link reftype=fn refid=afnr x=16 y=8 cx=16 cy=8.  
:link reftype=inform res=0345 x=0 y=8 cx=16 cy=8.
```

```
:link reftype=launch object='c:\os2\e.exe'
    data='c:\appsdir\tutor.dat' x=16 y=0 cx=16 cy=8.
:artlink.
```

:artwork. (Artwork)

Purpose

Identifies a bit map to be placed into the user's file.

Tag	Element	Attributes	End
:artwork.	Artwork	name=' ' align= linkfile=' ' runin fit	

Attributes

name='filename.ext'

Identifies the file containing the bit map (artwork). This attribute is required and must specify a complete file name.

align=left right center

Specifies how the artwork is to align with the current margins. It can be to the left, to the right, or centered.

linkfile='filename.ext'

Identifies the file with the link definitions. This file begins with **:artlink.** and ends with **:eartlink..** The **linkfile=** attribute enables you to link from whole or segmented bit maps. It can be omitted if the artwork file does not require links, or if the links are enclosed by **:artlink.** and **:eartlink.** immediately following the artwork tag.

runin

Specifies that the artwork is to be placed within the line of text. You enter **:artwork.** and its attributes in the line of text where you want the artwork to appear.

fit

Causes the artwork to fill the window in which it is displayed. If the user resizes the window, IPF redisplayes the graphic so that it fits the new window size.

When the initial size of the window is specified, the ratio between its width and height should be approximately the same as that of the graphic; otherwise, the artwork may appear distorted.

The **fit** attribute is most often used when artwork is to be displayed in a split window, where one window contains a bit map, and another contains text that is displayed beside the graphic.

If the artwork tag has **fit**, and you include text in the same window, the text will be displayed briefly, but will then be covered by the painting of the bit map in the window.

Description

Use **:artwork.** to include bit maps such as vectors and scanned images, in the text file. The artwork tag and its attributes enable you to merge whole or segmented bit maps and position them in the window. A bit map can be created by an application or by a bit-map editing tool, such as the Presentation Manager Icon Editor. A metafile can be created using a GPI function.

Conditions

- If a path name is not specified for either **name=** or **linkfile=**, IPF looks for the file in the current directory.
- If **linkfile=** is not specified, IPF looks for the artlink tag on the line immediately following the artwork tag.

- The artwork tag requires the **name=** attribute.

Example 1

This example shows how to include artwork that does not require a hypergraphic link. The artwork is to be placed within the line of text that contains the artwork tag.

Click on the `:artwork name='gopi.art' runin.` symbol to close the file.

Example 2

This example shows how to include artwork that fills the window in which it is displayed.

`:artwork name='c:\main\world.bmp' fit.`

:caution. (Caution)

Purpose

Alerts the user to a risk.

Tag	Element	Attributes	End
<code>:caution.</code>	Caution	<code>text= ' '</code>	<code>:ecaution.</code>

Attributes

`text= ' '`

Enables you to change CAUTION to different text.

Description

A caution message notifies the user of possible risks. It should precede the text to which it pertains so the user will see it first.

When `:caution.` is encountered, **CAUTION** appears on the screen, and the caution text is displayed on the next line. A blank line is inserted before the caution message.

Conditions

None

Example 1

```
:caution.
These berries are wild. Do not eat.
:ecaution.
```

Output

CAUTION:

These berries are wild. Do not eat.

Example 2

```
:caution text='Wild Berries:'.
These berries are wild. Do not eat.
:ecaution.
```

Output

WildBerries:

These berries are wild. Do not eat.

:cgraphic. (Character Graphic)

Purpose

Defines a character graphic.

Syntax

Tag	Element	Attributes	End
:cgraphic.	Character graphic		:ecgraphic.

Attributes

None

Description

Character graphics are those you create with an ASCII editor. The **:cgraphic.** tag indicates that a character graphic is to follow. Everything after the tag and before **:ecgraphic.** will be in a monospace font. A blank line is inserted before and after the graphic.

Conditions

Text that does not fit in the display area of a window is clipped.

Example

```
:cgraphic.
```

```
One
:hp4.Two:ehp4.
:ecgraphic.
```

Output

The following output is provided.

```
One
Two
```

:color. (Color)

Purpose

Changes the colors of the text and text background.

Syntax

Tag	Element	Attributes	End
<code>:color.</code>	Color	<code>fc=</code> <code>bc=</code>	

Attributes

`fc=`

Enables you to change the color of the text. Text following this attribute appears in the color specified. The values that can be specified are:

default
black
blue
red
pink
green
cyan
yellow
neutral
brown
darkgray
darkblue
darkred
darkpink
darkgreen
darkcyan
palegray

`bc=`

Enables you to change the background color of the text. The screen colors remain the same. The values that can be specified are the same as those for `fc=`.

Description

`:color.` and its attributes enable you to change the color of the text and the color of the text background. Colors set with this tag remain in effect until another color is specified or a heading definition is encountered.

To return to the system colors, use `fc=default` and `bc=default`.

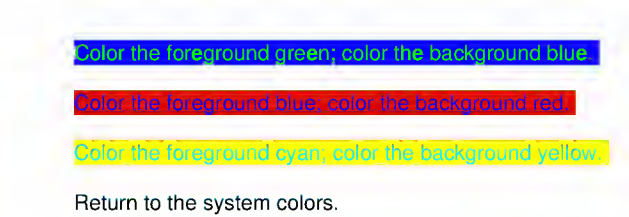
Conditions

None

Example

```
:sl.  
:color fc=green bc=blue.  
:li.Color the foreground green; color the background blue.  
.*  
:color fc=blue bc=red.  
:li.Color the foreground blue; color the background red.  
.*  
:color fc=cyan bc=yellow.  
:li.Color the foreground cyan; color the background yellow.  
.*  
:color fc=default bc=default.  
:li.Return to the system colors.  
:esl.
```

Output



:ctrl. (Control Area)

Purpose

Defines the contents of the control area.

Syntax

Tag	Element	Attributes	End
:ctrl	Control area	ctrlid= controls=' ' page coverpage	

Attributes

ctrlid= Specifies the identification value for the control area. The identification value can be either alpha or alphanumeric, and is referenced by the heading tag.

controls=' ' Specifies the identification values of the push buttons that you want included in the control area of a window. Push buttons are displayed in the order in which they are defined. The values that can be specified are:

Search	Specifies the Search push button. This push button displays a window that lets the user search for a word or phrase.
Print	Specifies the Print push button. This push button displays a window that lets the user print one or more topics.
Index	Specifies the Index push button. This push button displays an alphabetic list to the topics in the document.
Contents	Specifies the Contents push button. This push button displays the Contents window.
Esc	Specifies the Previous push button. This push button lets the user see information from an earlier request.
Back	Specifies the Back push button. This push button displays the previous page in the table of contents hierarchy.
Forward	Specifies the Forward push button. This push button displays the next page in the table of contents hierarchy.

Note: An identification value for the **Tutorial** push button is not provided because it is displayed automatically if a tutorial exists.

If you are defining your own push buttons, use **id=** attribute of the push button tag (**:pbutton.**). See [:pbutton. \(Push Button\)](#).

The identification values for the predefined push buttons are defined in the PMHELP.H file.

page Specifies that a set of push buttons is displayed in the control area of an IPF text window . You can use this attribute to override the

default set of push buttons that is displayed in the control area of an IPF text window.

coverage
Specifies the set of push buttons that is displayed in the control area of the coverage window . The control area in the coverage window is at the very bottom of a window. You can use this attribute to override the default set of push buttons that is displayed in the coverage window.

Example: The following tagging specifies the **Previous**, **Forward**, and **Back** push buttons display in the coverage window:

```
:ctrl ctrlid=new1 controls='ESC FORWARD BACK' coverage.
```

Description

The control area tag (**:ctrl.**) specifies where push buttons are displayed, and which push buttons you want displayed. You can display push buttons in the *control areas* of coverage window or an IPF text window.

The default control area for online documents and Help windows is the coverage page window, and the default push buttons that display are:

Online documents

Previous
Search
Print
Index
Contents
Back
Forward
Tutorial (only if a tutorial is available).

Help windows

Previous
Search
Print
Index
Tutorial (only if a tutorial is available).

You can define more than one control area with different sets of push button for an IPF text window; however, only one set of push buttons can be defined for the coverage window.

The default set of push buttons for an IPF text window can be overridden by defining a new default or by referring to the control area definition with the heading tag (see [:h1. through :h6. \(Headings\)](#)). For more information about push buttons, see [Push Buttons](#).

Conditions

- The control area tag (**:ctrl.**) must be enclosed within the control area definition tag (**:ctrldef.**) and associated end tag (**:ectrldef.**) (see [:ctrldef \(Control Area Definition\)](#)).
- The **:ctrl.** tag must follow all push button tags (**:pbutton.**) (see [:pbutton. \(Push Button\)](#)).

:ctrldef (Control Area Definition)

Purpose

Defines a control area.

Syntax

Tag	Element	Attributes	End
:ctrldef.	Control area definition	NONE	:ectrldef.

Attributes

None

Description

Use the **:ctrldef.** tag to define a control area and the contents of the control area. For tagging information about the control area of a window, see **:ctrl.** ([Control Area](#)).

Conditions

- This tag should follow the **:docprof.** tag.
- The following tags are embedded within the **:ctrldef.** and **:ectrldef.** tags.
 - **:pbutton.**
 - **:ctrl.**

:ddf. (Dynamic Data Formatting)

Purpose

Display dynamically formatted text in an application-controlled window.

Syntax

Tag	Element	Attributes	End
:ddf.	Dynamic data formatting	res=	

Attributes

res=

Associates a location in a window with a request for specific information. The value is an integer from 1 to 64 000.

Description

:ddf. indicates that the application will provide dynamically formatted data.

When IPF encounters **:ddf.**, it sends the message HM_QUERY_DDF_DATA to the OBJCOM window, and specifies the **res=** value. (The application identified the OBJCOM window by sending HM_SET_OBJCOM_WINDOW to IPF.) The OBJCOM code responds by initializing for dynamic data formatting and proceeding with a dynamic data-formatting routine, using dynamic data-formatting functions. For more information, see [Changing Help Information at Run Time \(DDF\)](#).

:dl. (Definition List)

Purpose

Identifies a list of terms and definitions.

Syntax

Tag	Element	Attributes	End
:dl.	Definition list	compact tsize= break=	:edl.
:dthd.	Definition-term heading		
:ddhd.	Definition-description heading		
:dt.	Definition term		
:dd.	Definition description		

Attributes

compact

Causes the list to be formatted without a blank line between each term and description. If you omit this attribute, a blank line is inserted.

tsize=10 | n

Defines the amount of space to allot for the terms and term headings. The default is 10 character units. If the value of **tsize=** exceeds the current size of the formatting area (the space between the current margins), the current formatting area size is assigned, and a warning message is issued.

break=none | fit | all

Controls the formatting of the terms and descriptions:

none	The description is on the same line as the term. If the length of the term exceeds the value specified by tsize= , the term extends into the description column, and the description starts one space after the term.
fit	The description is placed on the line below the term if the term is longer than the value specified by tsize= .
all	All descriptions are placed on the line below the term.

Conditions

- The term-heading tag (:dthd.) is paired with the description-heading tag (:ddhd.) and precedes the term and description tags (:dt. and :dd.).
- The term tag requires a description tag.

Example

```
:dl compact tsize=20.
:dthd.:hp2.Mammal:ehp2.
:ddhd.:hp2.Description:ehp2.
:dt.Florida Panther
:dd.Relative of the mountain lion or puma.
:dt.Key Deer
:dd.&odq.Toy&cdq. member of the whitetail deer family.
:dt.Manatee
:dd.Gentle giant sea cow.
:edl.
```

Output

Mammal	Description
Florida Panther	Relative of the mountain lion or puma.
Key Deer	"Toy" member of the whitetail deer family.
Manatee	Gentle giant sea cow.

:docprof. (Document Profile)

Purpose

Specifies the heading-level entries to be displayed in the Contents window.

Syntax

Tag	Element	Attributes	End
:docprof.	Contents window	toc=	
	entries		
	Global communications dynamic-link library	dll=' ' objectname=' ' objectinfo=' '	
	Push button support	ctrlarea=	

Attributes

toc=

Enables you to control the heading levels displayed in the table of contents. For example, if you want only heading levels 1 and 2 to appear, the tagging is:

```
:docprof toc=12.
```

If no **toc=** value is specified, heading levels 1 through 3 appear in the Contents window. Heading levels 4 through 6 appear as part of the text when the window is displayed.

If a heading tag also specifies a value for **toc=**, the heading-tag value overrides the **:docprof.** value until either the end of the file is reached, or another heading **toc=** value is encountered.

dll=' '

Specifies the communication dynamic-link library that IPF loads so that a communication object in the library can be used to alter the behavior of IPF. For author-defined push buttons, this is the communication object that will receive the HM_NOTIFY message. For a **Tutorial** push button, this is the communication object that will receive the HM_TUTORIAL message.

objectname=' '

Identifies the entry point of the communication object in the dynamic-link library. The value for this attribute is case sensitive.

objectinfo=' '

Identifies parameters to be passed to the object.

ctrlarea=

Defines the control areas in a window where you want to display push buttons. For more information on control areas in a window, see [Push Buttons](#).

Possible values are:

page	Identifies the control area within the IPF text window.
coverpage	Identifies the control area as the bottom of the coverpage window. This is the default value.
both	Specifies that you want a control area in both the IPF text window, and the coverpage window.
none	Specifies that you do not want a control area (that is, you do not want push buttons).

Description

:docprof. is placed at the beginning of the file. It follows the title tag (**:title.**), if a title is specified; otherwise, it follows the user-document tag (**:userdoc.**).

The **:docprof.** tag also provides the ability to alter the behavior of IPF by loading any dynamic-link modules specified in the **dll=** attribute. It is

possible to have multiple windows and multiple entry points within a dynamic-link module. There is no guarantee that the ACVP structure that is passed into the object will be maintained across multiple calls.

You also can use this tag to change the size and function of the coverpage and its client and control windows.

The **:docprof.** tag can be used to define the control area in a window where you want to display push buttons.

Conditions

None

Example

```
:userdoc.  
:title.  
:docprof toc=123 ctrlarea=none.  
:euserdoc.
```

Output

When the user selects the "+" icon in the Contents window, heading levels 1 through 4 are displayed in a tree-structured format. There are no push buttons because of **ctrlarea=none**.

:fig. (Figure)

Purpose

Identifies a figure.

Syntax

Tag	Element	Attributes	End
:fig.	Any text		:efig.

Attributes

None

Description

:fig. indicates that what follows is to be formatted exactly as it is entered. Text that exceeds the window area will be clipped.

The figure is displayed in proportional font, with a blank line preceding the text. Because proportional font is used, words will align, but letters and numbers may not.

Conditions

None

Example

```
:fig.  
Area   Number  Classification      Code  
  
JOB1  
  
      2      Full-time exempt      1A  
      4      Part-time exempt      1B  
      4      Full-time nonexempt    2A  
      1      Part-time nonexempt    2B
```

```
:efig.
```

Output

Area	Number	Classification	Code
JOB1			
	2	Full-time exempt	1A
	4	Part-time exempt	1B
	4	Full-time nonexempt	2A
	1	Part-time nonexempt	2B
	2	Supplemental	3A

:figcap. (Figure Caption)

Purpose

Specifies a figure title.

Syntax

Tag	Element	Attributes	End
:figcap.	Any text		

Attributes

None

Description

:figcap. is placed between :fig. and :efig.. The text of the caption goes on the same line as the tag, or on the next line.

Conditions

- Use :figcap. either immediately after :fig. or immediately before :efig..
- The text of the figure caption cannot contain tags or semicolons.

Example

```
:fig.
Area   Number  Classification      Code

JOB2

      5      Full-time exempt      1A
      1      Part-time exempt      1B
      3      Full-time non-exempt  2A
      1      Part-time non-exempt  2B
      1      Supplemental          3A

:figcap.Payroll Codes for Area JOB2
```

:efig.

Output

Area	Number	Classification	Code
JOB2			
	5	Full-time exempt	1A
	1	Part-time exempt	1B
	3	Full-time non-exempt	2A
	1	Part-time non-exempt	2B
	1	Supplemental	3A

Payroll Codes for Area JOB2

:font. (Font)

Purpose

Changes the font to the specified typeface, size, and code page.

Syntax

Tag	Element	Attributes	End
:font.	Fonts	facename= size= codepage=	None

Attributes

facename=

Defines the typeface name of the font. You can use any bit map font in the system.

This attribute is required. If **default** is specified, the font is reset to the default system font.

Notice that **facename=** values have initial capitals. These are required; otherwise, the IPF compiler will not recognize them as valid values. No error message is returned when an invalid value for **facename=** is encountered.

size=hx w

Defines the average character height and width, in *points*, of the Presentation Manager image font. (A point is a typesetting measure that is equal to approximately 1/72 of an inch.)

Following are the Presentation Manager image fonts available on all system-supported display adapters:

Face

Courier

Helv

Tms Rmn
System
Monospaced

System
Proportional

The **size=** attribute is required. If the value is set to 0x0, the font is reset to the default system font.

codepage=

Specifies the code page to be used. This is a three-digit number. Possible values are:

437 - U.S. IBM PC
850 - Multilingual
860 - Portuguese
863 - Canadian French
865 - Nordic

See [Country Code, Code Pages, and Language Parameters](#) for a list of countries and their code pages.

The **codepage=** attribute is optional. If no code-page value is specified, the code page of the active system process is used.

Description

:font changes the current font for the text within the current window. When a heading **tag** defining a new window is encountered, the font resets to the default system font.

You can make as many font changes within a window as you want. If you define highlighted phrases while a font tag is in effect, the highlighted text will be displayed in the font style corresponding to the specified typeface.

When you specify height and width values for a valid font name, you do not have to know the exact point values. If no match is found for a specified font size, IPF uses a "best fit" method to select the font. For example, suppose you specify:

:font facename=Helv size=20x12.

IPF selects "Helv 18x10" because it is the closest match.

:fn. (Footnote)

Purpose

Identifies a pop-up window.

Syntax

Tag	Element	Attributes	End
:fn.	Pop-up	id=	:efn.

Attributes

id=

Specifies the ID of the footnote. It is used in conjunction with the link tag (see [:link. \(Link\)](#)).

Description

The footnote tag encloses information that will be displayed in a pop-up window when the user selects a hypertext link to the information. Footnotes can appear within paragraphs, lists, highlighted phrases, and artwork.

Conditions

- Index entries are not valid within a footnote.
- The *id=* attribute is required.
- One footnote must end before another begins.
- A footnote cannot be linked from a child window.
- Information in a footnote cannot be returned as the result of a search.

Example

The following shows how to enter the footnote ID (here "ddrive") and provide a link to the footnote.

```
:fn id=ddrive.
```

The information you place here appears in the pop-up window as a footnote. For example, you could enter additional information about the disk drive in a footnote.

```
:efn.
```

To provide the link that allows the user to view the footnote pop-up, you could enter:

```
:p.Additional information about
:link refid=ddrive reftype=fn.disk drives:elink.
is available.
```

Output

When the information is displayed, *disk drives* is highlighted and clicking on *disk drives* pops up the footnote window.

:h1. through :h6. (Headings)

Purpose

Define window characteristics.

Syntax

Tag	Element	Attributes	End
:h1.-:h6.	Define cross references to internal and external sources.	res= id= name= global tutorial=' '	
	Define origin and size of windows with relation the primary window.	x= y= width= height=	
	Manage the display of information in multiple windows.	group= viewport clear	
	Define the user's control over the window.	titlebar= scroll= rules=	
	Restrict user retrieval of information.	nosearch noprint hide	
	Change heading levels that	toc=	

appear in the
Contents
window.

Define the ctrlarea=
control area of ctrlrefid=
a window for
displaying push
buttons.

Attributes

res=
id=
name=

Specify window identifiers.

If you are creating an .HLP file, **res=** is required and can be any integer from 1 through 64 000. However, if you are creating an .INF file (compiled by specifying the /INF parameter with the IPFC command), you can use **res=**, **name=** or **id=**. With **name=** and **id=**, you can include alphabetic characters. You cannot use these attributes if you plan to concatenate .INF files. Instead, you must use **res=**. For more information see [Concatenating .INF Files](#).

global

Indicates to IPF that the [window](#) can be a reference in an external database (another IPF .HLP or .INF file). A reference from one IPF database to another is made by specifying **reftype=hd** and **database='filename'** with the link tag.

tutorial=' '

Specifies the file name of the tutorial and causes the tutorial choice to be added to the help pull-down when the window is displayed. When the user selects **Tutorial**, the HM_TUTORIAL message specifying the file name of the tutorial is sent to the application. An example of the tagging follows:

```
:h1 tutorial='example.exe'.Test Window
```

x=
y=
width=
height=

Define the size and position of a window. The **x=** and **y=** attributes are values along the x and y axes; they define the origin of the window. The x axis runs horizontally from left to right, and the y axis runs vertically from bottom to top. The position where the values specified for **x=** and **y=** intersect is the the origin of the window. (The 0,0 intersection is the bottom left corner of the parent window.) From this location, width and height are measured. For more information about window coordinates, see [Defining Window Origin and Size](#).

Size and position attributes can be given in absolute, dynamic, or relative values:

Absolute value:

A number followed by a letter, which indicates the unit of measure:

c	(Characters): Average character width of the default system font.
x	(Pixels): Dependent on the display adapter in use.
p	(Points): Typesetting measure; equal to approximately 1/72 inch.

Relative value:

A number followed by the percent sign (%), indicating a percentage of the parent-window width or height.

Dynamic value:

A term indicating a window coordinate location that is dependent on the current size and position of the primary window:

center | left | right

For x= values: In the center of, flush left in, or flush right in the parent window.

center | top | bottom

For y= values: In the center of, at the top of, or at the bottom of the parent window.

Restrictions:

When defining window position and size, you cannot mix absolute values with dynamic or relative values for either of the following combinations of attributes:

- The x coordinate and the width
- The y coordinate and the height.

If no values for **x=** and **y=** are specified, the origin of the window is 0,0. If you specify values other than 0,0, you also must specify width and height values. Negative values for these attributes are not allowed.

group=
viewport
clear

The **group=** attribute enables you to assign the window a number from 1 through 64 000. This associates the window with a heading definition and the IPFC information that follows it. If you do not provide a number with **group=**, IPF assigns the number 0.

A group number can be assigned to a viewport by a heading or link definition. For example, suppose you have a group number specified in a link definition, and another in the heading that the link refers to. If a user action causes the link definition to be selected, the link group number overrides the heading group number. However, if the user selects the heading from either the Contents window or the Index window, the heading group number takes effect.

IPF searches among the open windows to find one with a number matching the one specified with **group=**. If no match is found, IPF opens a new window. If a match is found, the information associated with the group number is swapped with the information in the matched window.

The **viewport** attribute always opens a window. If you specify both **viewport** and **group=**, and a window with the specified group number is already open, IPF opens another window with the same group number. Thus, it is better that you do not specify the **viewport** attribute in a heading that will appear in the Contents window, unless you want your contents entries to always open separate windows.

The **clear** attribute causes IPF to close any open windows before opening a window to display the current window.

titlebar=yes | **sysmenu** | **minmax** | **both** | **none**
rules=border | **sizeborder** | **none**
scroll=horizontal | **vertical** | **both** | **none**

These attributes define Presentation Manager window controls and are used primarily when defining secondary windows. If none of these attributes are specified, the default is to open a window that has a title bar with title bar icon, hide button, maximize button; a sizing border; and vertical and horizontal scroll bars.

nosearch
noprint
hide

These attributes restrict information retrieval and are most often used in heading definitions for secondary windows.

The **nosearch** attribute in a secondary heading definition prevents the heading from being returned as an entry in the search-results window. This does not mean the secondary window is not searched. It is, however, only the primary heading definition that is returned. When the user selects the primary heading definition, the contents of the second window are displayed as part of the primary-window composition.

The **noprint** attribute in a secondary heading definition prevents the contents of a secondary window from being printed as a separate entity. Instead, secondary windows are printed as part of their primary window. The contents of secondary windows are printed in the order in which the link definitions are listed in the primary window.

When used in secondary heading definitions, **nosearch** and **noprint** merely prevent duplication of output (search results or printed copy). When used in regular heading definitions, they prevent retrieval of the information by the user. The only exception to this condition is if the user selects **This section** for either printing or searching.

The **hide** attribute prevents a heading level from appearing in the Contents window; however, there must be at least one heading level that is not hidden.

toc=

Specifies heading levels that are to be entries in the Contents window. When this attribute is encountered in a heading tag, the specified heading levels override any levels specified by **toc=** of the document-profile tag (**:docprof.**) until either the end-of-file is reached or another **toc=** attribute is encountered in a heading tag. If no document-profile tag exists, the heading levels that appear in the Contents window are levels 1, 2, and 3. The **toc=** attribute must be greater than zero.

ctrlarea=

Specifies which control area in a window you want to display push buttons. When this attribute is encountered in a heading tag, it overrides the **ctrlarea** attribute specified by **:docprof.** Possible values are:

page	Identifies the control area as the IPF text window.
none	Specifies that you do not want a control area.

For example, if your document consisted of 100 windows, and you wanted only one window to display push buttons in the IPF text window, you would tag your source file as follows:

```
:docprof ctrlarea=none.
.
.
.
:h1 ctrlarea=page.One Window
```

ctrlrefid=

Refers to the identification value (**id=**) specified by the control area tag (**:ctrl.**). This attribute specifies which control area you want to display for this heading. This attribute is used to override the default control area (the coverage window).

Note: Be careful when using heading tags to define a control area for split windows. A control area cannot be defined in the secondary window heading tag of a split window. You must define the control area (the coverage window) in the primary window heading tag.

:hdref. (Heading Reference)

Purpose

Places a hypertext link to a heading. The text displayed within the link is the word "Reference".

Syntax

Tag	Element	Attributes	End
:hdref.	Heading Reference	res= refid=	

Attributes

res=

Resource identification number of the heading to link to.

refid=

ID of the heading to link to.

:hide. (Hide)

Purpose

Controls display of IPF text and graphics to meet conditions set by the IPF_KEYS= environment variable.

Syntax

Tag	Element	Attributes	End
:hide.	Hide	key=	:ehide.

Attributes

key=' '

Defines the key that enables a user to view hidden information. You can specify one or more key names. Enclose each key name within apostrophes. When specifying more than one key name, insert a plus (+) sign after each name.

Text entered between the **:hide.** and **:ehide.** tags is only displayed when the **key=**' ' attribute matches the entry specified by the user. Use the OS/2 environment variable SET IPF_KEYS= to specify the key name identified for the hidden information.

set in

If this attribute is not specified, the information identified by the hide tag is displayed.

Description

:hide. enables you to determine what text and graphics will be displayed within a window. This function is useful when you want to tailor the information you give to users; for example, if you want to display levels of information on the basis of a user's system configuration, you assign each level a value with the **key=** attribute. When a topic containing hide tags is selected for viewing, IPF will look for an environment variable called IPF_KEYS= to determine what level of information to show the user. If a match is found, the information within the hide tags is displayed; otherwise, the information is hidden from view.

The hide tag affects the display of compiled information. You can hide lines of text within the window, a word or a phrase within a line, or you can hide an instruction to display a bit map, as in the following example.

```
:hide key='level1'.  
:artwork name='mybitmap.bmp'.  
:ehide.
```

If the user's environment does not contain the key to display the hidden information, IPF wraps the text from the last character or formatting instruction on the line preceding **:hide.** to the line following **:ehide.**

In some situations, the same user may need to view more than one level of hidden information. This can be accomplished by setting the IPF_KEYS= to concatenated values; for example:

```
SET IPF_KEYS=LEVEL1+LEVEL2
```

Take care that a window view does not contain an orphan tag. For example, you do not want to hide the information following a list item, unless you have alternate information to display, based on the setting of a key. In the case of an ordered list, which generates sequential numbers, you would not include a list item in the hidden information, unless it is the last item in the list.

Conditions

- You cannot nest one set of hide tags within another.
- You cannot include a heading tag that has a **res=** attribute within a set of hide tags.
- You cannot set IPF_KEYS= on a session basis.

Example

Suppose the following source has been compiled as part of a help library file:

```
:h1 res=001.Installation Procedure  
:ol.  
:li.  
:hide key='usera'.  
Instruction for User A.  
:ehide.  
:hide key='userb'.  
Instruction for User B.  
:ehide.  
:li.  
Shut down the system from the Desktop.  
:li.  
Press Ctl+Alt+Del to restart the system.  
:eol.
```

If the user's environment includes the setting, IPF_KEYS=USERA, the following is displayed:

1. Instruction for User A.
2. Shut down the system from the desk top.

3. Press Ctrl+Alt+Del to restart the system.

:hp1. through :hp9. (Highlighted Phrase)

Purpose

Emphasize text by changing the font style or foreground color.

Syntax

Tag	Element	Attributes	End
:hp <i>n</i> .	Highlighting	None	:ehp <i>n</i> .

Description

Highlighted-phrase tags are useful for emphasizing words and phrases within text.

Font styles that are displayed for highlighted phrases correspond to the typeface currently being used by IPF. To change from the default system typeface to other typefaces, use :font. When you use either the example tag (:xmp.), the character-graphics tag (:cgraphic.) or the table tag (:table.), the system monospace typeface is displayed.

Input

```
:sl compact.  
:li.:hp1.Highlighted phrase 1 looks like this.:ehp1.  
:li.:hp2.Highlighted phrase 2 looks like this.:ehp2.  
:li.:hp3.Highlighted phrase 3 looks like this.:ehp3.  
:li.:hp4.Highlighted phrase 4 looks like BLUE.:ehp4.  
:li.:hp5.Highlighted phrase 5 looks like this.:ehp5.  
:li.:hp6.Highlighted phrase 6 looks like this.:ehp6.  
:li.:hp7.Highlighted phrase 7 looks like this.:ehp7.  
:li.:hp8.Highlighted phrase 8 looks like RED.:ehp8.  
:li.:hp9.Highlighted phrase 9 looks like PINK.:ehp9.  
:esl.
```

System Default Font Output

```
Highlighted phrase 1 looks like this.  
Highlighted phrase 2 looks like this.  
Highlighted phrase 3 looks like this.  
Highlighted phrase 4 looks like this.  
Highlighted phrase 5 looks like this.  
Highlighted phrase 6 looks like this.  
Highlighted phrase 7 looks like this.  
Highlighted phrase 8 looks like this.  
Highlighted phrase 9 looks like this.
```

Conditions

You cannot nest highlighted-phrase tags.

:i1. and :i2. (Index)

Purpose

Place topics into the index.

Syntax

Tag	Element	Attributes	End
:i1.	Primary entry	id= global roots=' ' sortkey=' '	
:i2.	Secondary entry	refid= global sortkey=' '	

Attributes

id=

Provides a cross-reference identifier for the secondary index tag (:i2.). This attribute is optional and only valid when used with the primary index tag (:i1.).

global

Specifies that the index entry can appear in any MIndex Object depending on where you install the file and how each MIndex Object is set up. Entries also appear in the component index. This attribute is only used in Help windows. Online document cannot use this attribute.

roots='root words '

Specifies a list of root words that act as index entries to specified topics. These root words are associated with words defined with the index-synonym tag (:isyn.). Root words can contain alphabetic and numeric characters, which can be entered in uppercase or lowercase. When entering a string of words, insert a blank space between each word, and enclose the string within apostrophes.

Root words do not appear in the index, so are not viewed by the user, and need not be translated. They are used to create a link between the primary index tag and the index-synonym tag. To enable the user to search for an index entry, use the index-synonym tag to map the root words associated with the entry to synonyms.

sortkey='sortkey-text '.index-text

Specifies a character string that is used for sorting the entry in the index, and another character string that is displayed for the index entry.

The *sortkey-text* character string determines where this entry is placed in the index. The *index-text* character string is displayed for the index entry.

refid=

Provides a reference to the text associated with the primary index tag.

Description

You use the primary and secondary index tags to provide index entries to the information. The attributes associated with each index tag enable you to define related information. Index entries can be used throughout the file, but cannot be placed within a footnote.

The text of the index entry must be on the same line as the tag, and cannot contain other tags. The entry for each primary index entry within the window must be unique. That is, you cannot provide duplicate index entries within the same window. Secondary index entries must refer to an identifier specified for a primary index entry.

When the user selects *Help index* from the Help menu, an Index window is displayed for the help interface. When the user selects *Index* from the Options menu, an Index window is displayed for the online information interface. If the user enters a synonym that matches a root word, the index topics listed for the root word are displayed. When the user selects the **Master Help Index** object from the Desktop, it opens to display an alphabetic list of entries within a spiral bound notebook. For more information, see [Master Help Index and Glossary](#)

Conditions

- Index entries cannot appear in a footnote.
- When referencing the **:i1.** tag, use the **global** attribute on both the **:i1.** and **:i2.** tag.

Example 1

This example shows how to tag your file to include primary and secondary index entries.

```
:i1 id=del.delete
:i2 refid=del.directories
:i2 refid=del.files
```

Output

The index will include the following entry:

```
delete
  directories
  files
```

Example 2

This example shows a file with the index-synonym tag (**:isyn.**) and the **roots=** attribute.

```
:h1 id=copy03.Help for Copying
:isyn root=copy.copy copying duplicate duplicating
:isyn root=book.book manual draft manuscript
:isyn root=folder.folder folders document documents
:i1 roots='copy folder'.Copying a document
:i1 roots='book folder'.Test procedures
:p.When copying a file from the current directory to a new
directory, specify the following:
:ul.
:li.The file name
:li.The target directory
:li.The new file name and extension.
:eul.
```

Output

The index-synonym tag creates the following synonym table:

Root word	Synonym words
copy	copy copying duplicate duplicating
book	book manual draft manuscript
folder	folder folders document documents

The **roots=** attribute points to the root words, "copy" and "folder," and the list of associated synonyms. For example, if the user searches for "copy" or "folder," the "Copying a document" entry appears because "copy" and "folder," identified by the index **roots=** attribute, match the entries listed for the index synonym **root=** attribute.

A search for the synonym "duplicate" lists "Copying a document" as one of the index choices. A search for the synonym "manual" lists "Test procedures" as an index choice, and a search for "document" lists both "Copying a document" and "Test procedures."

Example 3

This example shows how to specify a sort key to change the location of the entry in the index.

```
:i1 sortkey='point sizes'.changing fonts
:i1.program header
:i1.parameter list
:i1.preface
```

Output

The index will include the entry "changing fonts" at the location where the term "point sizes" would appear in the sorting sequence of the index, as follows:

```
parameter list
changing fonts
preface
program header
```

Example 4

This example is for a simple **Master Help Index** object entry for conceptual information about batch files.

:i1 **global**.batch files, creating

When referring to an **:i1**. tag, use the **global** attribute in both the **:i1**. and **:i2**. tags. For example:

```
:i1 id=copy global.copying
:i2 refid=copy global.help topics
:i2 refid=copy global.document topics
```

When the IPF compiler encounters **global** attributes, it creates an alphabetic list, which can then be accessed by selecting **Master Help Index** object from the Desktop.

:icmd. (Index Command)

Purpose

Identifies the help window that describes a command.

Syntax

Tag	Element	Attributes	End
:icmd.	Index command	<i>external command string</i>	

Attributes

external-command-string

Specifies the command for which help is being defined. The text can contain no other tags.

Description

The help information for a command is assumed to be in the help window in which the index-command tag (**:icmd.**) is defined. If the help window provides help for more than one command, an index-command tag should be defined within the heading tag for each command.

The same external command string cannot be specified in more than one index-command tag of an index file; that is, only one help window can be designated as describing a command.

If the compiler finds the same external command string more than once (either from the same or different help windows), the duplicate occurrences are discarded, and a warning message is issued.

Note: The association with entry field and command names is a programming task. In addition, the application developer must define the field with which command windows are to be associated as a command entry field. For more information about programming a command entry field, see [Command Entry Field](#).

Conditions

:icmd. must follow a heading tag or another index tag.

Example

```
:h1 id=xhlp.Help for Copying
:icmd.Copying
:h1 res=129.Deleting Files
:icmd.Delete
```

Output

At execution time, the index entries enable the compiler to process command helps, create a list of commands for which help is available, and display the help window defined for any of those commands.

:isyn. (Index Synonym)

Purpose

Identifies synonyms and word variations for the help keywords.

Syntax

Tag	Element	Attributes	End
:isyn.	Index synonyms	root=	

Attributes

root=

Links synonyms and variations of words specified in a primary index tag.

To establish a link, specify the same word as specified in the **roots=** attribute of the primary index tag. Then add a period, repeat the root word, and add the list of synonyms and variations, separated by blanks. For example, assume that the value specified for the **roots=** attribute of the primary index tag is "copy." The entry for the index-synonym tag could be:

```
:isyn root=copy.copy copying duplicate duplicating
```

The words entered in the synonym list enable the user to search for terms that may not be in the Index list, and still receive the appropriate help. Lowercase and uppercase characters are treated the same.

Description

:isyn. begins a list of synonyms or variations of a word specified by a primary index tag. The compiler uses this list to build a table that serves as a link to the primary index tags. Synonyms determine the topic entries displayed when the user enters words for a search of the index. The compiler matches the entered words with words in the table and links to the topics to be displayed.

The index-synonym tag can be placed within any window that contains related index entries identified by the index tag. The synonyms defined in a window can relate to many topics, and thus to many windows.

Synonyms defined with this tag do not appear in the index.

Conditions

A root word can contain only alphabetic and numeric characters.

Example

```
:h1 id=copy03.Help for Copying
:isyn root=copy.copy copying duplicate duplicating
:isyn root=folder.folder folders document documents
:il roots='copy folder'.Copying a document
:p.When copying a file from the current directory to a new
directory, specify the following:
:ul.
:li.The file name
:li.The target directory
```

```
:li.The new file name and extension
:eul.
```

Output

The index-synonym tag creates the following synonym table:

Root word	Synonym words
copy	copy copying duplicate duplicating
folder	folder folders document documents

The **roots=** attribute points to the root words, "copy" and "folder," and the list of associated synonyms. If the user searches for "copy" or "folder," the words will be displayed because of the matches between the **roots=** attribute of the primary index tag and the **root=** attribute of the index-synonym tag. However, a search for the synonym "duplicate" returns "Copying a document" as an index choice.

:li. (List Item)

Purpose

Identifies an item within a list.

Syntax

Tag	Element	Attributes	End
:li.	List item		

Attributes

None

Description

The format of the list items depends on the type of list: ordered, unordered, or simple. For example, if the list is an ordered list, a number precedes each list item. If the list is an unordered list, a bullet precedes each item. See [:ol. \(Ordered List\)](#), [:sl. \(Simple List\)](#), and [:ul. \(Unordered List\)](#) for more information.

Conditions

None

Example

```
:p.To remove a diskette&colon.
:ol.
:li.Open the drive door.
:li.Remove the diskette.
:li.Put the diskette in a safe place.
:eol.
```

Output

To remove a diskette:

1. Open the drive door.
2. Remove the diskette.
3. Put the diskette in a safe place.

:lines. (Lines)

Purpose

Turns formatting off.

Syntax

Tag	Element	Attributes	End
:lines.	Lines	align=	:elines.

Attributes

align=left right center

Places the entered lines to the left in the window, to the right, or in the center.

Description

:lines. specifies that the following text is to be formatted exactly as it is entered. The attributes enable you to align the text within the window. Text that is too long for the window is clipped.

Proportional fonts are used for formatting, so the text may not be displayed exactly as entered.

Conditions

None

Example 1

This example aligns text to the left.

```
:lines align=left.  
The warehouse contained:  
 12 desks  
 28 chairs  
 15 lamps  
 39 typewriters  
 11 pictures  
:elines.
```

Example 2

This example aligns text to the right.

```
:lines align=right.  
The warehouse contained:  
 12 desks  
 28 chairs  
 15 lamps  
 39 typewriters  
 11 pictures  
:elines.
```

Output

The following output is provided.

Example 1

The warehouse contained:
12 desks
28 chairs
15 lamps
39 typewriters
11 pictures

Example 2

The warehouse contained:
12 desks
28 chairs
15 lamps
39 typewriters
11 pictures

:link. (Link)

Purpose

Activates a link to additional information.

Syntax

Tag	Elements	Attributes	End
:link.	Link to more information	reftype= res= refid= database=' ' object=' ' data=' '	:elink.
	Automatic linking	auto viewport dependent split child group=	
	Define window position and size	vpx= vpy= vpcx= vpcy=	
	Define window controls	titlebar= scroll= rules=	

Attributes

reftype=
Defines the type of link. Possible values are **hd**, **fn**, **launch**, and **inform**.
reftype=hd

Links to a heading. The heading definition (or an overriding definition in the link) causes its information to be displayed in the current window or another window. The integer value of **refid=** identifies the ID of the heading. If the heading is in an external IPF database, its file name is specified with the **database=** attribute.

In the following example, selection of the hypertext link causes the external database, EDITOR.HLP, to be loaded, and the heading with the ID of 001 to be displayed.

```
:link reftype=hd refid=001.  
      database='editor.hlp'.  
Editing Functions
```

:elink.

The heading definition in the external database must contain the **global** attribute. If the link to the file cannot be resolved, the hypertext phrase in the link will not be highlighted.

reftype=fn

Links to a footnote. Its contents are displayed in a pop-up window in the current window. The **refid=** attribute specifies the ID of the footnote.

Restriction: A split window cannot contain a link to a footnote.

reftype=launch

Starts a Presentation Manager program. The file name of the program is specified with the **object=** attribute. Any parameters to the program are specified with **data=**. In the following example, the hypertext link starts the System Editor and opens the file, MYFILE, for editing.

```
:link reftype=launch
      object='c:\os2\se.exe'
      data='myfile'.
Start Editor
:elink.
```

Restriction: You can only use alpha-numeric characters in the **data=** field when using the **reftype=launch** attribute.

reftype=inform

Causes a message to be sent to the application. The **res=** attribute is required and is an integer value that directs the application to perform some application-specific function. When using this attribute, do not use **:elink..** For example:

```
:link reftype=inform res=1000 auto.
```

auto
viewport
dependent
split
child
group=

With the **auto** attribute, you can define any of the link types described above, with the exception of a footnote link, as an automatic link.

The automatic-link definition follows a heading definition and is activated as soon as a reference to the heading definition is made. The reference can be made by the user selecting an IPF window entry (for example, the Contents window), or by a hypertext or hypergraphic link.

Restriction: Linking automatically to an external database is not possible.

Following are the automatic-link actions that can be specified, and the attributes used:

- **Open a secondary window** when the heading that contains the link is referred to:

```
auto reftype=hd viewport dependent res=
```

Note the inclusion of the **dependent** attribute. Usually, the information in an automatic window is dependent on the information in its secondary window. Specifying **dependent** causes an automatic window to close when the user closes the window of the secondary that contains the automatic link.

- **Open secondary windows** when the heading of the primary window that contains the links is referred to:

```
auto reftype=hd split res=
```

Restriction: The primary heading cannot contain text or graphics; only links to its secondary headings. For more information, see [Split Windows](#).

- **Start a Presentation Manager program** when the heading that contains the link is referred to:

```
auto reftype=launch object= data=
```

- **Send the application a message** when the heading that contains the link is referred to:

```
auto reftype=inform res=
```

To display more than one window on the screen, you must assign a unique group number to each window with the **group=** attribute. This attribute can be specified with **:link.** or the heading tag. For more information about group numbers, see [Displaying Multiple Windows](#).

The **child** attribute opens the panel being linked to as a child of the panel in which the link is located. This means:

- The child panel is clipped to fit inside the parent panel and cannot be moved outside the parent panel.
- The child panel is always "on top" of the parent panel and the parent cannot be moved on top of the child (the parent can still receive mouse clicks and keystrokes).
- The child panel is closed when the parent panel is closed.
- The child panel is minimized when the parent panel is minimized and restored when the parent is restored.
- When the child panel is minimized by itself, its minimized icon is displayed inside the parent panel instead of on the coverpage.

The **child** attribute opens the help panel as a child.

vpx=
vpy=
vpcx=
vpcy=

Define the size and position of the window. Any values specified by these attributes override size and position values specified by the attributes in a heading tag. (See [:h1.](#) through [:h6.](#) ([Headings](#)) for details about these attributes.)

Restriction: These attributes are not valid for positioning or sizing a footnote ([:fn](#)) window.

titlebar=yes | **sysmenu** | **minmax** | **both** | none
scroll=horizontal | vertical | **both** | none
rules=border | **sizeborder** | none

Define window controls. Any values specified by these attributes override window-control values specified by the attributes in a heading tag. (See [:h1.](#) through [:h6.](#) ([Headings](#)) for details about these attributes.)

When **titlebar=yes** is specified, the window displays a titlebar without the system menu symbol, the hide button, and the maximize button.

:lm. (Left Margin)

Purpose

Sets the left margin of the text.

Syntax

Tag	Element	Attributes	End
:lm.	Left margin	margin=	

Attributes

margin=

Specifies where the left margin of the text is to begin. To set the margin for the current line, specify a number greater than the position of the cursor. For example, to set the left margin to 15, begin the left margin tag before space 15. Otherwise, the margin becomes effective on the next line.

Note: When counting character spaces, you are actually counting average character widths.

Description

Use the left-margin tag and the right-margin tag (:rm.) to specify the boundaries of the text in the window.

When the text window is sized, the text area adjusts from the right to fit within the specified margin boundaries; that is, the right margin adjusts to the new window size. The left margin remains constant. If the window is sized smaller than the specified margins, the margins remain the same, and the text area is reduced to one character space.

You can place multiple margin tags in your file. The margins specified remain effective until they are reset. If no margin value is specified, the default is 1.

Conditions

None

Example

This example shows the use of both margin tags.

```
:p.
:rm margin=10.
:lm margin=20.This text begins 20 spaces to the
right of the left window border and ends 10 spaces to the
left of the right window border.
All text is aligned as specified
by the margin values. :lm margin=5.Here the left margin
is changed to 5. Because this margin tag begins
more than 5 spaces on the line, the margin specified
becomes effective on the following line, and the text
begins 5 spaces from the left window border.
The right margin remains unchanged.
```

Output

This text begins 20 spaces to the right of the left window border and ends 10 spaces to the left of the right window border. All text is aligned as specified by the margin values. Here the left margin is changed to 5. Because this margin tag begins more than 5 spaces on the line, the margin specified becomes effective on the following line, and the text begins 5 spaces from the left window border. The right margin remains unchanged.

:lp. (List Part)

Purpose

Identifies an explanation within a list.

Syntax

Tag	Element	Attributes	End
:lp.	List part		

Attributes

None

Description

:lp. can be entered anywhere within the list. The text following the tag starts at the left margin of the current list item. It is not numbered or lettered. Using the list-part tag does not interrupt the sequence of the list.

Conditions

None

Example

```
:p.To remove a diskette:
:ol.
:li.Open the drive door.
:lp.Before removing the diskette, make sure all drive activity
has stopped.
:li.Remove the diskette.
:li.Put the diskette in a safe place.
:eol.
```

Output

To remove a diskette:

1. Open the drive door.

Before removing the diskette, make sure all drive activity has stopped.

2. Remove the diskette.
3. Put the diskette in a safe place.

:note. (Note)

Purpose

Starts a note.

Syntax

Tag	Element	Attributes	End
:note.	Note	text= ' '	

Attributes

text= ' '

Enables you to change the name of the note.

Description

:note. identifies a single-paragraph note. When the tag is encountered, a blank line is inserted, and the note starts at the left margin with **Note:** followed by two blank spaces. The start of another tag ends the note, so no end tag is needed.

When the tag is used within a list, the note aligns with the text of the items within the list.

Use the **text= ' '** attribute to give the note a specific name.

Conditions

None

Example 1

```
:note.
This text appears within a note.
The word :hp2.Note:ehp2. aligns
with the text that precedes it.
```

Output

The following output is provided.

Note: This text appears within a note. The word **Note** aligns with the text that precedes it.

Example 2

```
:note text='Text note:'.
The name of this note is :hp2.Text note:ehp2..
The name of the note replaces
the word :hp2.Note:ehp2.. The name of the note
aligns with the text that precedes it.
```

Output

Text note: The name of this note is **Text note**. The text for the note replaces the word **Note**. The name of the note aligns with the text that precedes it.

:nt. (Note)

Purpose

Starts a note that can have multiple paragraphs.

Syntax

Tag	Element	Attributes	End
:nt.	Note	text=' '	:ent.

Attributes

text=' '
Enables you to change the name of the note.

Description

:nt. starts a new paragraph with **Note:** followed by two blank spaces and the first line of the text. The second and succeeding lines of text align with the first line, to the right of **Note:**.

Notes can be placed within lists and paragraphs; however, unlike the **:note.** tag, **:nt.** requires an end tag.

You can use the **text=' '** attribute to assign a specific name to the note.

Conditions

None

Example

```
:nt.
Use this tag to include paragraphs in a note.
You also can use it within paragraphs and lists.
:p.End this tag before you begin another
note tag.
:ent.
```

Output

Note: Use this tag to include paragraphs in a note. You also can use it within paragraphs and lists.

End this tag before you begin another note tag.

:ol. (Ordered List)

Purpose

Starts a sequential list of items or steps.

Syntax

Tag	Element	Attributes	End
:ol.	Ordered list	compact	:eol.

Attributes

compact

Causes the list to be formatted without a blank line between each list item. If you omit **compact**, a blank line appears between each list item.

Description

:ol. indicates the start of an ordered list. Items in the list are entered with the list-item tag (**:li.**). The output is an indented list with each item numbered. Use the list-part tag (**:lp.**) for paragraphs within the list.

Ordered lists can be nested or imbedded within other lists. When this is done, the first list has sequential numbers at the left margin, and the nested list has sequential letters indented two spaces. After the second list, the number-letter sequence repeats for each successive ordered list.

Be sure to end each list with the end-list tag.

Example

```
:p.To remove a diskette:
:ol.
:li.Open the drive door:
:ol compact.
:li.Remove two screws.
:li.Lift the door.
:eol.
:li.Remove the diskette.
:li.Put the diskette in a safe place.
:eol.
```

Output

To remove a diskette:

1. Open the drive door:
 - a. Remove two screws.
 - b. Lift the door.

2. Remove the diskette.
3. Put the diskette in a safe place.

:p. (Paragraph)

Purpose

Starts a new paragraph.

Syntax

Tag	Element	Attributes	End
:p.	Paragraph		

Attributes

None

Description

Each paragraph identified by a paragraph tag formats as an unindented block of text. Paragraphs placed within a list align with the text of the list. When paragraphs are placed within a note, the text of the paragraph aligns with the text of the note.

Conditions

None

Example

```
:p.Paragraph tags cause a blank line before the text.
When placed within a list or note, the text of the paragraph
aligns with the text of the list or note.
:ul.
:li.Paragraph tags
:p.Paragraph tags are flexible and can be used
with most tags.
:li.Note tags
:p.Note tags can include paragraphs.
:eul.
```

Output

Paragraph tags cause a blank line before the text. When placed within a list or note, the text of the paragraph aligns with the text of the list or note.

- Paragraph tags
Paragraph tags are flexible and can be used with most tags.
- Note tags
Note tags can include paragraphs.

:parml. (Parameter List)

Purpose

Starts a two-column list of parameter terms and descriptions.

Syntax

Tag	Element	Attributes	End
:parml.	Parameter list	tsize= break= compact	:eparml.
:pt.	Parameter term		
:pd.	Parameter definition		

Attributes

tsize=	Specifies the space allocated for the parameter term. The default is 10 character units.
break=all fit none	Controls the formatting of the parameter terms and descriptions: break=all Causes the description to begin on the line below the parameter term, next to the space allocated by tsize= . This is the default. break=fit Causes the parameter description to begin on the same line as the term, if the term has fewer characters than specified by tsize= . If the term has more characters, the description begins on the line below the term. break=none Causes the description to begin on the same line as the term. If the term has more characters than specified by tsize= , it continues into the description area. The description starts one space after the end of the term.
compact	Causes the list to be formatted without a blank line between each list item. If you omit compact , a blank line appears between each item.

Description

Parameter lists are similar to definition lists; they define terms and descriptions that format in two columns. The elements of the parameter-list tag are the parameter-term tag (:pt.) and the parameter-description tag (:pd.). The term tag identifies the term, and the definition tag identifies the description.

Parameter lists can occur anywhere in text; you can nest them within other lists, and you can nest other lists within parameter lists.

Conditions

- Each parameter-term tag requires a parameter-description tag.
- Each parameter list requires an end-parameter-list tag.

Example

```
:parml compact tsize=10 break=none.  
:pt.Tree  
:pd.Plant life in forest  
:pt.Orange  
:pd.Fruit on tree  
:pt.Cow
```

```
:pd.Animal on farm
:eparm1.
```

Output

Tree	Plant life in forest
Orange	Fruit on tree
Cow	Animal on farm

:pbutton. (Push Button)

Purpose

Defines author-defined push buttons.

Syntax

Tags	Element	Attributes	End
:pbutton.	Author-defined push buttons	id= res= text=' '	

Attributes

id= Specifies the identification value for a push button that you define. The identification value can be alpha or alphanumeric. This identification value is referenced by the control area tag (:ctrl.).

res= Specifies the *resource* identification value for a push button that you define. This value is returned with the HM_NOTIFY and HM_CONTROL messages and can be any integer greater than 256 (0 to 256 are reserved for use by IPF).

text= '' Specifies the text for the push button that you define. Define the mnemonic for the push button by placing the tilde (~) character before the mnemonic character. For example:

```
:pbutton id=xmp res=300 text='~Example'.
```

Note: Make sure the mnemonic you specify for author-defined push buttons does not conflict with the mnemonics of the predefined set of push buttons, or with any of IPF's shortcut keys. See [:ctrl. \(Control Area\)](#) for a description of the control area tag (:ctrl.) and a list of the predefined push buttons and their associated mnemonics.

Description

Use the push button tag (:pbutton.) to define author-defined push buttons. For more information, see [Author-Defined Push Buttons](#).

:pd. (Parameter Description)

Purpose

Starts the description for a parameter term in a parameter list.

Syntax

Tag	Element	Attributes	End
:pd.	Parameter	description	

Attributes

None

Description

The text that follows **:pd.** describes the term identified by **:pt.**. The description formats in the right column, as defined by the values of **tsize=** and **break=**. For a description of **:parml.**, see [:parml. \(Parameter List\)](#).

A parameter list can have multiple parameter-term and parameter-description tags; however, each term tag requires a description tag.

Conditions

- The parameter-description tag follows the parameter-term tag.
- The parameter-description tag is valid only within a parameter list.

Example

```
:parml compact tsize=15 break=all.  
:pt.Tree  
:pd.Plant life in forest  
:pt.Orange  
:pd.Fruit on tree  
:pt.Cow  
:pd.Animal on farm  
:eparml.
```

Output

Tree	Plant life in forest
Orange	Fruit on tree
Cow	Animal on farm

:pt. (Parameter Term)

Purpose

Identifies a term in a parameter list.

Syntax

Tag	Element	Attributes	End
:pt.	Parameter	description	

Attributes

None

Description

The term identified by **:pt.** formats in the left column. The **:pt.** tag requires a parameter-description tag (**:pd.**); the description formats in the right column.

Conditions

- The parameter-term tag requires a parameter-description tag.
- The parameter-term tag precedes the parameter-description tag.
- The parameter-term tag is valid only within a parameter list (for a description of **:parml.**, see [:parml. \(Parameter List\)](#)).

Example

```
:parml compact tsize=15 break=all.  
:pt.Tree  
:pd.Plant life in forest  
:pt.Orange  
:pd.Fruit on tree  
:pt.Cow  
:pd.Animal on farm  
:eparml.
```

Output

Tree	Plant life in forest
Orange	Fruit on tree
Cow	Animal on farm

:rm. (Right Margin)

Purpose

Sets the right margin of the text.

Syntax

Tag	Element	Attributes	End
:rm.	Right margin	margin=	

Attributes

margin=

Enables you to indicate the number of character spaces from the right border of the window the text is to end. For example, **margin=60** means that the text is to end 60 spaces from the right border.

Note: When counting character spaces, you are actually counting average character widths.

Description

Use **:rm.** with the left-margin tag (**:lm.**) to specify the boundaries of the text in the window. The left-margin tag specifies where the text is to

start, and the right-margin tag specifies where it is to end.

You can enter margin tags at the beginning of the line of text or while you are entering the text. Margin tags that begin the line of text cause text on that line and the following lines to align with the values specified. Margins set while you enter text become effective on the current line or on the next line, depending on where the margin tag begins. For example, to set the right margin to 60 (that is, 60 spaces before the right border of the window), begin the right-margin tag at least 60 spaces to the left of the right border. When the file is displayed, the text entered after the margin tag aligns to the value specified on that line.

If the margin tag is started after the specified boundary, the margin becomes effective on the next line.

When the text window is sized, the text area adjusts from the right to fit within the specified margin boundaries; that is, the right margin adjusts to the window size. The left margin stays the same. If the window is sized smaller than the specified margins, the margins remain the same, and the text area is reduced to one character space. If no value is specified for **margin=**, the default for the right margin is 1.

You can place multiple margin tags in your file. The specified margins remain effective until they are reset.

Example

```
:lm margin=1.
:rm margin=44.
:p.In this example, the left margin is 1. The right
margin is 44. The margins are set before the text;
therefore, when the file is displayed, the text
formats according to the margins set.
The text begins at space 2 and ends 44 spaces before
the right window border. If the margin specified is
less than the current cursor position on the screen,
the margins set become effective on the following
line. For example, if the current cursor position is
60 spaces to the left of the right window border and
you set the right margin to 50, the margin is
effective on the current line. However, if the right
margin is set to 65, the margin becomes effective
on the next line.
:p.
:lm margin=5.
:rm margin=60.Here the left margin is set to 5
and the right margin is set to 60. This means that
the left margin begins 5 spaces to the right of the
left border. The right margin ends 60 spaces to the
left of the right border.
```

Output

In this example, the left margin is 1. The right margin is 44. The margins are set before the text; therefore, when the file is displayed, the text formats according to the margins set. The text begins at space 2 and ends 44 spaces before the right window border. If the margin specified is less than the current cursor position on the screen, the margins set become effective on the following line. For example, if the current cursor position is 60 spaces to the left of the right window border and you set the right margin to 50, the margin is effective on the current line. However, if the right margin is set to 65, the margin becomes effective on the next line.

Here the left margin is set to 5
and the right margin is set to
60. This means that the left
margin begins 5 spaces to the
right of the left border. The
right margin ends 60 spaces to
the left of the right border.

:sl. (Simple List)

Purpose

Starts a nonsequential list of items.

Syntax

Tag	Element	Attributes	End
:sl.	Simple list	compact	:esl.

Attributes

compact

Causes the list to be formatted without a blank line between each list item. If you omit **compact**, a blank line appears between each item.

Description

:sl. identifies items that do not require a sequential listing. Items in a simple list are not indented and do not have bullets, hyphens, or dashes preceding them. Simple lists can be nested within other lists. When nested, a simple list is indented four spaces to the right of the left margin of the list that contains it. Each list requires an end-list tag.

The simple-list tag requires the list-item tag (:li.) to identify items in the list. You can use the list-part tag (:lp.) to include paragraphs in the list.

Conditions

None

Example

```
:p.Bring the following for lunch:
:sl.
:li.Fruit, for example:
:sl compact.
:li.An apple
:li.An orange
:li.A pear
:li.A banana
:esl.
:li.Sandwich
:li.A drink, for example:
:sl compact.
:li.A soda
:li.Juice
:li.Milk.
:esl.
:esl.
```

Output

Bring the following for lunch:

Fruit, for example:

An apple
An orange
A pear
A banana

Sandwich

A drink, for example:

A soda
Juice
Milk.

:table. (Table)

Purpose

Formats information as a table.

Syntax

Tag	Element	Attributes	End
:table.	Tables	cols=' ' rules= frame=	:etable.
:row.	Rows	None	None
:c.	Columns	None	None

Attributes

cols=' '
Specifies the width, in character spaces, of each column; for example: **cols**='10 15 20'.

rules=
Specifies whether the table will have horizontal and vertical rules. Following are the possible values and meanings:

both	Horizontal and vertical rules
horiz	Horizontal rules only
vert	Vertical rules only
none	No rules

Note: The default is **both**.

frame=
Specifies whether the table will have borders. Following are the possible values and meanings:

rules	A horizontal line at the top and bottom of the table
box	A box around the table
none	No borders.

Note: The default is **box**.

The **:row.** tag specifies the start of each row in the table. The **:c.** tag specifies the text for each column entry in the table. The text provided with the **:c.** tag is formatted within the column. However, if a single word is longer than the specified width of the column, the word will be clipped.

Example

The following defines a table with three columns and two rows. The width of each column is 15, 20, and 25 character spaces.

```
:table cols='15 20 25' rules=both frame=box.  
:row.  
:c.Row 1 Col 1  
:c.Row 1 Col 2  
:c.Row 1 Col 3  
:row.  
:c.Row 2 Col 1
```

```
:c.Row 2 Col 2
:c.Row 2 Col 3
:etable.
```

Output

Row 1 Col 1	Row 1 Col 2	Row 1 Col 3
Row 2 Col 1	Row 2 Col 2	Row 2 Col 3

:title. (Title)

Purpose

Provides a name for the online document.

Syntax

Tag	Element	Attributes	End
:title.	Title		

Attributes

None

Description

The text that follows **:title.** provides a name for the online document. The title of an online document can contain up to 47 characters, including spaces and blanks. If the title exceeds 47 characters, the IPF Compiler displays an error message.

When you display the online document, the title appears on the title line of the main window. The title is limited to one line. Word wrapping does not occur in the title of an online document.

Conditions

Use the **:title.** tag only for the title of an online document. Do not use it for online help windows.

Example

```
:userdoc.
:title.Using the Information Presentation Facility
:h1 res=100.Creating an Index
:p.This section shows you how to create index entries.
:euserdoc.
```

Output

When you compile this file, "Using the Information Presentation Facility" is displayed on the title line of the main window of the online document.

"Creating an Index" is listed as an entry in the contents window. If you select "Creating an Index," the window with this heading and the accompanying text is displayed in the text information area, overlaying the contents window.

:ul. (Unordered List)

Purpose

Starts a list of nonsequential items.

Syntax

Tag	Element	Attributes	End
:ul.	Unordered list	Compact	:eul.

Attributes

compact

Causes the list to be formatted without a blank line between each list item. If you omit **compact**, a blank line appears between each item.

Description

:ul. indicates the start of a list of items that do not require sequential listing. The list-item tag (:li.) identifies the items within the list. The list-part tag (:lp.) is used to include paragraphs within the list.

Unordered list items are indented, and a bullet (lowercase "o") precedes each item. Unordered lists can be nested within other lists. If placed within an ordered list or a simple list, the nested list will be indented four spaces, and each item will be preceded by a bullet. If placed within another unordered list, the nested list will be indented four spaces, and each item will be preceded by a dash.

Conditions

None

Example

```
:p.Before leaving for the day, remember to:
:ul.
:li.Turn off the computer.
:li.Turn off the lights:
:ul compact.
:li.Ceiling
:li.Desk
:eul.
:li.Secure all equipment.
:eul.
```

Output

Before leaving for the day, remember to:

- Turn off the computer.
- Turn off the lights:
 - Ceiling
 - Desk
- Secure all equipment.

:userdoc. (User Document)

Purpose

Identifies the source file that is to be compiled.

Syntax

Tag	Element	Attributes	End
<code>:userdoc.</code>	User document		<code>:euserdoc.</code>

Attributes

None

Description

:userdoc. must be the first tag in the source file. It signals the compiler to begin compiling the tagged text that follows. All other tags that define how the text is to be formatted follow this tag.

The end-user-document tag (**:euserdoc.**) identifies the end of the tagged text and the end of the source file. It must be the last tag in the source file.

Conditions

None

Example

```
:userdoc.  
.br/>.br/>.br/>.br/>:euserdoc.
```

:warning. (Warning)

Purpose

Alerts the user of a risk or possible error condition.

Syntax

Tag	Element	Attributes	End
<code>:warning.</code>	Warning	<code>text=' '</code>	<code>:ewarning.</code>

Attributes

`text=' '`
Enables you to give a specific name to the warning notice.

Description

A warning notice alerts the user to a possible risk, such as an error condition in the system. It should appear before the text that it discusses.

Use the `text=` attribute to provide a specific name for the warning notice.

Conditions

None

Example 1

```
:warning.  
The disk contains bad sectors.  
:ewarning.
```

Output

The following output is provided.

Warning: The disk contains bad sectors.

Example 2

```
:warning text='Bad disk:'.  
The disk contains bad sectors.  
:ewarning.
```

Output

Bad disk The disk contains bad sectors.

:xmp. (Example)

Purpose

Turns formatting off.

Syntax

Tag	Element	Attributes	End
:xmp.	Example		:exmp.

Attributes

None

Description

Text entered between `:xmp.` and `:exmp.` is formatted as entered, in a monospace font. The text is indented two spaces from the left margin of the window. Lines that are too long to fit within the window are clipped.

Conditions

- An example cannot be placed within another example.
- An end-example tag is required.

Example

```
:xmp.
#define INCL_WIN
#include <os2.h>

MRESULT EXPENTRY MyObject(PACVP pACVP, PCH ObjectInfo)
{
    HWND hwndMyACVP;          /* Handle to the application-controlled */
                              /* window that this procedure creates */
    .
    .
    .
}
:exmp.
```

Output

```
#define INCL_WIN
#include <os2.h>

MRESULT EXPENTRY MyObject(PACVP pACVP, PCH ObjectInfo)
{
    HWND hwndMyACVP;          /* Handle to the application-controlled */
                              /* window that this procedure creates */
    .
    .
    .
}
```

Symbols

This chapter discusses the symbols you can use to display special characters that you may want to include in your file. Symbols can be used to specify characters that are not on the keyboard. Each symbol represents a single character. When tagging your file to include symbols, begin each symbol with an ampersand (&) and end the symbol with a period (.). For example, to place a square bullet () in a file, you would enter:

```
&sqbul.
```

Symbols are case sensitive; that is, uppercase characters produce different symbols than lowercase characters. Therefore, when tagging the file to include a symbol, enter the tag for the symbol exactly as it is shown in the symbols table.

Note: All symbols in the following table are also in the APSYMBOL.APS file. This file is in the \TOOLKIT\IPFC directory and can be edited with any text editor; however, some National Language code pages require a different symbols file. See [National Language Support \(NLS\)](#), for a list of these files.

Symbol	Symbol Name	Character
&aa.	a acute	á
&ac.	a circumflex	â
&ae.	a umlaut	ä
&Ae.	A umlaut	Ä
&ag.	a grave	à
æ.	ae ligature	æ
Æ.	AE ligature	Æ
&Alpha.	Alpha	Α

&.	ampersand	&
&and.	and	^
&angstrom.	angstrom	Å
&ao.	a overcircle	ā
&Ao.	A overcircle	Ā
&apos.	apostrophe	'
&bx2022.	ASCII code 185	
&bx2020.	ASCII code 186	
&bx0022.	ASCII code 187	
&bx2002.	ASCII code 188	
&bx2200.	ASCII code 200	
&bx0220.	ASCII code 201	
&bx2202.	ASCII code 202	
&bx0222.	ASCII code 203	
&bx2220.	ASCII code 204	
&bx0202.	ASCII code 205	
&bx2222.	ASCII code 206	
&asterisk.	asterisk	*
&atsign.	at sign	@
&bslash., &bsl.	back slash	\
&Beta.	Beta	Β
&bxas., &bxbj.	box ascender	
&bxcr., &bxcj.	box cross	
&bxde., &bxtj.	box descender	
&bxh.	box horizontal	
&bxle., &bxlj.	box left junction	
&bxll.	box lower-left	
&bxlr.	box lower-right	
&bxri., &bxrj.	box right junction	
&bxul.	box upper-left	
&bxur.	box upper-right	
&bxv.	box vertical	
&cc.	c cedilla	ç
&Cc.	C cedilla	Ç
&caret.	caret symbol	^
&cdq.	close double quote	"
&cdqf.	close French double quote	»
&csq.	close single quote	'
&comma.	comma	,
&colon.	colon	:
&dash.	dash	-

°ree., °.	degree	°
÷	divide	÷
&dollar.	dollar sign	\$
&dot.	dot	·
&ddarrow.	down arrow	↓
&ea.	e acute	é
&Ea	E acute	É
&ec.	e circumflex	ê
&ee.	e umlaut	ë
&eg.	e grave	è
&emdash.	em dash	—
&endash.	en dash	–
&eq., &equals., &eqsym.	equal sign	=
&xclm., &xclam.	exclamation point	!
&fnof.	function of	f
>sym., >.	greater than	>
&house.	house	ℎ
&hyphen.	hyphen	–
&ia.	i acute	í
&ic.	i circumflex	î
&ie.	i umlaut	ï
&ig.	i grave	ì
&inve.	inverted exclamation mark	¡
&invq.	inverted question mark	¿
&llarrow.	left arrow	←
&lahead.	left arrowhead	↰
&lbrace., &lbrc.	left brace	{
&lbracket. &lbrk.	left bracket	[
&lpar. , &lparen.	left parenthesis	(
&lnot.	logical not	¬
&mdash.	em dash	—
&minus.	minus sign	−
&mu.	mu	μ
&ndash.	en dash	–
&nt.	n tilde	ñ
&Nt.	N tilde	Ñ
&lnot., ¬sym.	not symbol	¬
&numsign.	number sign	#
&oa.	o acute	ó
&oc.	o circumflex	ô

&og.	o grave	ò
&oe.	o umlaut	ö
&Oe.	O umlaut	Ö
¼.	one fourth	¼
½.	one half	½
&odq.	open double quote	"
&odqf.	open French double quote	«
&osq.	open single quote	`
&percent.	percent	%
&per.	period	.
&plus.	plus sign	+
&plusmin., &pm.	plusminus	±
&lsterling.	pound sterling	£
&rbl.	required blank	
&arrow.	right arrow	"
&rahead.	right arrowhead	"
&rbrace., &rbr.	right brace	}
&rbracket., &rbrk.	right bracket]
&rpar., &rparen.	right parenthesis)
&semi.	semicolon	;
&box14.	shaded box 1/4 dots	
&box12.	shaded box 1/2 dots	
&box34.	shaded box 3/4 dots	
&slash., &slr.	slash	/
&BOX.	solid box	
&BOXBOT.	solid box bottom half	
&splitvbar.	split vertical bar (piping symbol)	
&sqbul.	square bullet	
².	superscript 2	²
&tilde.	tilde	~
&ua.	u acute	ú
&uc.	u circumflex	û
&ug.	u grave	ù
&ue.	u umlaut	ü
&Ue.	U umlaut	Ü
&us.	underscore	_
&aus.	underscored a	ª
&ous.	underscored o	º
&ye.	y umlaut	ÿ

Note: The left and right arrows appear as left and right arrowheads. The ASCII escape sequences do not permit true left and right arrows.

Compiler Error Messages

This chapter lists the error messages sent by the IPF compiler.

Description and Format of Error Messages

There are three types of error messages:

- *Warning Level 1* . They are the most severe.
- *Warning Level 2* . They are moderately severe.
- *Warning Level 3* . They are the least severe.

These error messages have the following format.

```
<C:\IPFC\YOURFILE.IPF:999> 124: Invalid tag in footnote [ ]

                                Optional error
                                information.
                                Tag, filename, etc.

                                Error message

                                Error code

                                Line number in source file
                                where error occurred

                                Filename of source file

                                Drive and path of source file
```

Warning Level 1 Messages

101	Invalid document body
	Explanation: No :userdoc./:euserdoc. match.
	This is a fatal error that will cause the compiler to stop executing.
102	Invalid tag syntax
103	Missing hypertext information
104	Cannot hide parent head level
	Explanation: Preceding head level must be hidden.
105	Illegal context for tag
	Explanation: Tags are not properly matched, a tag is used incorrectly, or a tag is placed incorrectly.
106	List start tag missing-tag ignored

107 List end tag not matched-tag ignored
108 Ignoring unmatched tag
109 Cannot open file

Explanation: SYSTEM ERROR. Filename or path is incorrect, file doesn't exist, or other system problem.

110 No id for this reference
112 No id for this footnote
113 No text found in tag
114 Page is too big

Explanation: Panel is too big. Maximum size is 16 000 words and punctuation marks. (Note maximum size is language dependent.)

116 Cannot create panel(s)

This is a fatal error that will cause the compiler to stop executing.

117 Duplicate text in tag
118 Duplicate root word
119 Duplicate tag in tag file
120 Ignoring text before :h1. tag
121 Invalid head level

Explanation: Head levels are not in consecutive order.

Example: If :h1. and :h3. are used and :h2. is missing, this error will occur.

122 Definition term or header not matched
123 Unexpected end of file

Explanation: This may be caused by an ending tag not being found, a corrupted or truncated source file, or a control-Z character found before the true end of file.

This is a fatal error that will cause the compiler to stop executing.

124 Invalid tag in footnote
125 Not enough memory

Explanation: SYSTEM ERROR. Close some applications to free some memory.

This is a fatal error that will cause the compiler to stop executing.

126 Cannot free memory

Explanation: SYSTEM ERROR. System could not free memory.

This is a fatal error that will cause the compiler to stop executing.

127 Cannot read file

Explanation: SYSTEM ERROR. Source file may be corrupted.

This is a fatal error that will cause the compiler to stop executing.

129 Document is too big - unique words exceed 16 000

This is a fatal error that will cause the compiler to stop executing.

130 A DT tag is not defined
131 A PT tag is not defined
132 Cannot write to a file

Explanation: SYSTEM ERROR. File system is full, out of disk space, diskette is write protected, etc.

This is a fatal error that will cause the compiler to stop executing.

133	Attribute not defined
134	Tag not defined
135	Invalid bitmap format
	Explanation: File is not a valid PM format bitmap file.
140	Invalid country code, or codepage
	This is a fatal error that will cause the compiler to stop executing.
141	Invalid language code
	This is a fatal error that will cause the compiler to stop executing.
143	No valid COLS specification was given
144	Ignoring invalid tag in table cell
145	Ignoring text before :c. tag
146	Extra cells will be placed in next table row
147	Missing ELINK tag inserted at end of table cell
148	Total table width exceeds limit of 250 characters
149	Cannot reopen. File is already opened
	Explanation: SYSTEM ERROR.
150	Document has no vocabulary
	This is a fatal error that will cause the compiler to stop executing.
151	No res for this reference
152	Duplicate tag in source file
153	Document has no visible table of contents entry
	This is a fatal error that will cause the compiler to stop executing.
154	Truncating table entry
	This is a warning to tell the user when the text for a table cell is too long for the size of the cell.

Warning Level 2 Messages

201	Invalid tag
202	Invalid attribute
203	Invalid symbol
	Explanation: Invalid APS symbol; period missing after the APS symbol, symbol specified is not in the APSYMBOL.APS file, invalid APSYMBOL.APS file.
204	Invalid macro
205	Text too long in tag
	Explanation: Heading and index tags have a maximum of 150 characters.
206	Token is bigger than expected.
	Explanation: Maximum length of token is 255 characters. This error could be caused by a missing end period or quote character.

207	Invalid attribute value
208	Missing tag
209	Attribute not matched
210	Text too long in macro expansion
Explanation: Maximum 255 characters.	
211	Total number of fonts exceeds the limit of 14
212	Sub index cannot be global without global main index
213	Invalid nest

Warning Level 3 Messages

301	Ignoring attribute
302	Duplicate ID
Explanation: Cannot specify the same ID in the same panel or index.	
303	Duplicate symbol in symbol file
304	Duplicate res number
305	Parent panel cannot have its own text
306	Missing panel text in head level tag
307	Missing footnote text in :fn. tag

Enabling Help for Applications

While running an application the user sometimes requires help. For example, the user may need assistance in making a choice, recalling the name of an application command or the use of a function key, or locating information.

Using IPF, you can develop a user interface that provides general help for application windows, and contextual help for fields within windows.

Implementing the IPF user interface when creating helps for an application requires two different development efforts:

- Developing the programming code that communicates with IPF and the Presentation Manager to display help windows.
- Developing a library of help information that IPF refers to in response to a user request.

This section will concentrate on the first development effort: writing the programming code that enables communication between IPF and the Presentation Manager.

Developing the Application Code

Use the following steps to develop the application code that adds help to your application.

1. Set up the help table and help subtable, and include the help constants defined in PMHELP.H.
2. Initialize the HELPINIT structure with a call to DosLoadModule.
3. Create a help instance.
4. Associate the help instance with the application window chain.
5. Respond to messages for menu bar choices.

6. End the help instance.

The following sections describe how to implement each of these steps.

Setting Up the IPF Help Tables

Two table structures in application memory or in resource files (.RC file-name extension), identify window resources in the IPF library. The help table associates each application window with its corresponding help subtable and the window identifier of its extended help window. The help subtable associates each entry field, menu item and push button within an application window with the window identifier (ID) of its help window. The address of the help table is passed to the application during initialization of the IPF initializing structure (HELPINIT).

When the user requests help on a field, menu bar, or push button in the application window, IPF uses the help subtable associated with the field to find the window ID of the contextual help window for the field. The help subtable also can store optional entries relating to application-specific information.

The maximum size of the help table is 64KB. The number of help subtables is limited to 16 000.

IPF supports two methods of defining help tables and help subtables. They can be allocated in memory, or they can be defined as resources. In either case, the information passed to IPF is identical.

Defining Help Tables in Memory

By defining help tables and subtables in memory, you can dynamically change a single entry in the help table. You can add a new window ID to be associated with a field, or add fields that are to be associated with existing windows.

After the help table structure is initialized, the application can pass IPF the address in memory of the new help table, either by sending the HM_CREATE_HELP_TABLE message from its window procedure, or by calling WinCreateHelpTable.

When defining help tables in memory, the data structures in PMHELP.H are used. The help table contains the structure for each application window. This structure holds the following information:

- Application window ID
- Address of the window's subtable
- Window ID of the window's extended help window

These entries are integers. The last entry in the list contains a NULL for each entry type, to indicate the end of the list. The following is an example of a help table for an application.

```
HELPSUBTABLE      table1, table2, table3,
                  table4, table5;

HELPTABLE         helpTableEntry [] =
{
    APP_WIND_1,    &table1, idExtHelp1,
    APP_WIND_2,    &table2, idExtHelp2,
    APP_WIND_3,    &table3, idExtHelp3,
    APP_WIND_4,    &table4, idExtHelp4,
    APP_WIND_5,    &table5, idExtHelp5,
    0,             NULL,     NULL
};
```

The help subtable contains the structure defined in the PMHELP.H file for each application window. This structure holds the following information for each field in the application window:

- Field ID from which user requests help
- Window ID of the contextual help window associated with the field
- Optional, application-specific integers.

The last entry in the structure contains the word length for each field entry. The minimum number of words is two, which is the default. The following is an example of a help subtable for an application window that has six fields.

```
HELPSUBTABLE      HelpSubTable [] =
{
    2,
```

```

FIELD_ID_1, IDRES_HELP1,
FIELD_ID_2, IDRES_HELP2,
FIELD_ID_3, IDRES_HELP3,
FIELD_ID_4, IDRES_HELP4,
FIELD_ID_5, IDRES_HELP5,
FIELD_ID_6, IDRES_HELP6,
0, 0
};

```

Defining Help Tables as Resources

If help tables are defined as resources, they can be bound to the application's executable file, or they can reside in a dynamic link library (DLL).

If help tables are defined as resources in a dynamic link library, the application must call `DosLoadModule` to load the DLL before it calls `WinCreateHelpInstance`. When the application calls `WinCreateHelpInstance`, it passes the handle to the DLL and the resource ID of the help table in the `HELPMINIT` structure.

The application can load a new help table residing in the DLL by either sending the `HM_LOAD_HELP_TABLE` message from its window procedure, or by calling `WinLoadHelpTable`. The application passes the handle to the DLL and the resource ID of the new help table.

A `HELPTABLE` resource contains a `HELPIITEM` entry for each application window, dialog, and message box for which help is provided.

Each entry of a `HELPTABLE` resource contains:

- `HELPIITEM` keyword
- Application window ID
- ID of the `HELPSUBTABLE` resource
- Window ID of the extended help window.

A `HELPSUBTABLE` resource contains an entry for each item that can be selected in an application window. Each of these items is assumed to be a child window of the application window identified in the `HELPTABLE` resource. The `HELPSUBTABLE` should contain a single `SUBITEMSIZE` and a `HELPSUBITEM` for each control, child window, and menu item.

Each entry of a `HELPSUBTABLE` resource contains:

- `HELPSUBITEM` keyword
- Field ID
- Window ID of the field's help window (corresponds to the resource number specified in the heading tag of the help-text window)
- Optional, application-defined integers.

The integer ID of the field can be a control, menu item, or message box ID. The ID specified must be unique within the table. An ID of hex FFFF is reserved for use by IPF.

The optional integers value allows the writer of the resource script file to append additional integers to the end of each `HELPSUBITEM` for application-specific use.

The `SUBITEMSIZE` keyword is used to identify the size in words of each `HELPSUBITEM`. All entries must be the same length. If this value is specified, it must be greater than or equal to 2. If this value is not specified, it defaults to 2.

All referenced `HELPSUBTABLE` resources must reside in the same `.RES` file as the `HELPTABLE` resource.

Following is an example of `.RC` source file for defining a `HELPTABLE` and its related `HELPSUBTABLE` resources.

```

HELPTABLE TABLE_1
BEGIN
HELPIITEM parentwindow1, SUBTABLE_1,
    extendedhelppanel1
HELPIITEM parentwindow2, SUBTABLE_2,
    extendedhelppanel2
END

HELPSUBTABLE SUBTABLE_1
[SUBITEMSIZE subitemsize1]
BEGIN
    HELPSUBITEM FIELD_ID1, helppanel1 [,
        integer1, ...n]
    HELPSUBITEM FIELD_ID2, helppanel2 [,
        integer1, ...n]

```

```

END

HELPSUBTABLE SUBTABLE_2
[SUBITEMSIZE subitemsize2]
BEGIN
    HELPSUBITEM FIELD_ID3, helppanel3 [,
        integer1, ...n]
    HELPSUBITEM FIELD_ID4, helppanel4 [,
        integer1, ...n]
END

```

Initializing the HELPINIT Structure

Before you call WinCreateHelpInstance, you must allocate memory for and initialize the HELPINIT structure. This structure defines values that IPF needs to create the help instance. Some of the values can be changed by your application after initialization.

The HELPINIT structure and the help table structures referred to by IPF during help processing are contained in the PMHELP.H file. The PMHELP.H file also contains the error codes returned in the event of an unsuccessful call. You include this file in your source code by using the INCL_WINHELP define statement. The following shows the HELPINIT structure.

```

typedef struct _HELPINIT /* hinit */
{
    ULONG          cb;
    ULONG          ulReturnCode;
    PSZ            pszTutorialName;
    PHELPTABLE     phtHelpTable;
    HMODULE         hmodHelpTableModule;
    HMODULE         hmodAccelActionBarModule;
    ULONG          idAccelTable;
    ULONG          idActionBar;
    PSZ            pszHelpWindowTitle;
    ULONG          fShowPanelId;
    PSZ            pszHelpLibraryName;
} HELPINIT;

```

Following are descriptions of the HELPINIT structure fields.

Field Name	Description
cb	The length of the initialization structure. This value can be use to identify the version of IPF being used.
ulReturnCode	The IPF return code
pszTutorialName	A pointer to a tutorial name, if one exists. If this value is NULL, either the application help interface does not include a tutorial, or the tutorial is referenced from a help window. If this value is not NULL, IPF provides a Tutorial choice in the help pull-down and adds a push button to the control area. If the user selects the Tutorial choice in the pull-down, IPF sends the HM_TUTORIAL message to the application so that it can start the tutorial.
phtHelpTable	If the help table is in memory this ia a pointer to the help table. If the help table is a resource, this is the resource ID ORed with 0xFFFF0000.
hmodHelpTableModule	The name of the resource file that indexes the dynamic link library that contains the help table and its corresponding subtables. If the help table is not being accessed through a dynamic link library, this value is 0.
hmodAccelActionBarModule	The name of the dynamic link library that contains the modified menu bar. If you do not have a modified menu bar, this value is 0.
idAccelTable	The name of the accelerator table if you are using a modified menu bar; otherwise, this value is 0.
idActionBar	The identity of the menu bar (action bar) template. If you are not modifying the menu bar, this value is 0.

pszHelpWindowTitle	A pointer to the name of the title for the main help window. This name can be changed after initialization by sending the message HM_SET_HELP_WINDOW_TITLE.
fShowPanelId	<p>A flag used to append the window ID to the beginning of the help window title in the title bar of the help window. If this flag is set to CMIC_SHOW_PANEL_ID, the window IDs are displayed. If this flag is set to CMIC_HIDE_PANEL_ID or to 0, the window IDs are not displayed.</p> <p>This flag is useful during the development stages of the help interface.</p> <p>After initialization, this flag can be toggled with the HM_SET_SHOW_PANEL_ID message.</p>
pszHelpLibraryName	<p>The help library names of the .HLP files containing the help windows. These .HLP files are created by the IPF compiler. When IPF needs to search for a help window, it looks for these library names in the path set by the HELP environment variable. If IPF cannot find a library name in this path, it then searches the current directory.</p> <p>After initialization, help library names can be specified with the HM_SET_HELP_LIBRARY_NAME message. If multiple libraries are specified, library names must be separated by a blank space.</p>

The following example shows a help facility being initialized. Notice that hmodAccelActionBarModule, idAccelTable, and idActionBar have values set to 0; this is because this example uses a standard menu bar.

```

VOID HelpInit (VOID)
{
    HELPINIT hini;

    /* if we return because of an error, Help will be disabled */
    fHelpEnabled = FALSE;

    /* initialize help init structure */
    hini.cb = sizeof (HELPINIT)
    hini.ulReturnCode = 0L;

    hini.pszTutorialName = (PSZ)NULL /*if tutorial added, add name here*/

    hini.phtHelpTable = (PHELPTABLE)MAKELONG(JIGSAW_HELP_TABLE, 0xFFFF);
    hini.hmodHelpTableModule = (HMODULE)0;
    hini.hmodAccelActionBarModule = (HMODULE)0;
    hini.idAccelTable = 0;
    hini.idActionBar = 0;

    if (!WinLoadString (habMain,
        (HMODULE)0,
        IDS_HELPWINDOWTITLE,
        HELPLIBRARYNAMELEN,
        (PSZ)szWindowTitle))

    {
        MessageBox (habMain, IDS_CANNOTLOADSTRING, MB_OK | MB_ERROR, FALSE);
        return;
    }

    hini.pszHelpWindowTitle = (PSZ)szWindowTitle;

    /* if debugging, show panel ids; else, don't */
#ifdef DEBUG
    hini.fShowPanelId = CMIC_SHOW_PANEL_ID;
#else
    hini.fShowPanelId = CMIC_HIDE_PANEL_ID;
#endif

    if (!WinLoadString (habMain,
        (HMODULE)0,
        IDS_HELPPLIBRARYNAME,
        HELPLIBRARYNAMELEN,
        (PSZ)szLibName))
    {
        MessageBox (habMain, IDS_CANNOTLOADSTRING, MB_OK | MB_ERROR, FALSE);
        return;
    }

    hini.pszHelpLibraryName = (PSZ)szLibName;

```

Creating the Help Instance

The WinCreateHelpInstance call passes the HELPINIT structure defined in the PMHELP.H include file to the Presentation Manager. WinCreateHelpInstance returns a handle to the help instance, which you must store in a HWND variable for use with the rest of the application programming interface (API) function calls associated with IPF.

IPF responds to the WinCreateHelpInstance call by installing its help hook and initializing for help processing.

The following shows how a help instance is created.

```
/* Creating help instance */
hwndHelpInstance = WinCreateHelpInstance (hMain, &hini);

if (!hwndHelpInstance || hini.ulReturnCode)
{
    MessageBox (hwndFrame,
                IDS_HELPLOADERROR,
                MB_OK | MB_ERROR,
                TRUE);
    return;
}
```

Associating the Instance with the Window Chain

After an application creates a help instance, it must associate the instance with the application window chain by calling WinAssociateHelpInstance. IPF uses the active window handle passed by this call to index into the help table to find the help window that should be displayed for the application window.

An IPF instance can be associated with any application window that has a frame. Once the association of an IPF instance with the application window chain is made, help can be requested for any application window in the chain.

The following shows how a help instance is associated with the application window chain.

```
/* associate the help instance with the main frame */
if (!WinAssociateHelpInstance (hwndHelpInstance, hwndFrame))
{
    MessageBox (hwndFrame,
                IDS_HELPLOADERROR,
                MB_OK | MB_ERROR,
                TRUE);
    return;
}

/* IPF is successfully initialized; set flag to TRUE */
fHelpEnabled = TRUE;
```

Ending the Help Instance

To end the current help instance, the application calls WinDestroyHelpInstance, passing the handle of the help instance that is to be ended.

The parameter *hwndHelpInstance* is the handle to the IPF instance returned from the WinCreateHelpInstance call.

The following shows how a help instance is terminated.

```
VOID DestroyHelpInstance (VOID)
{
    if (hwndHelpInstance)
```



```

{
    WinDestroyHelpInstance (hwndHelpInstance);
}
}

```

Responding to Messages for Menu Bar Choices

IPF communicates with the active window. This communication is accomplished with messages. The application may need to do some of its own processing in response to these messages.

Processing "Using help" Requests

When the user selects "Using help" from the help pull-down menu, a WM_COMMAND is sent to the application's window procedure.

If the application has created its own "Using help" window, it responds by sending the HM_REPLACE_USING_HELP message with the help-window ID. If the application chooses to use the "Using help" window provided by IPF, it responds by sending the HM_DISPLAY_HELP with NULL in both parameters.

Current CUA guidelines recommend applications use "Using help"; however, IPF continues to support the "Help for help" window.

Processing a "Keys Help" Request

When the user selects "Keys help" from the help pull-down, an HM_KEYS_HELP message is sent by the application to IPF. In response, IPF sends an HM_QUERY_KEYS_HELP message to the application. The application returns the window ID of the keys help window.

Processing Help Requests for a Child Window

In the Presentation Manager, parent and child windows are active at the same time. Therefore, when a help instance is associated with a window, its descendants are included in the association. However, only the parent window is the active help window.

Note: Do not confuse child windows with dialog, message boxes, and other windows which the application may own but are actually children of the desktop.

For IPF to process help requests for a child window, an application must send IPF HM_SET_ACTIVE_WINDOW messages to set the active help window. Until this happens, IPF continues to satisfy help requests for the child window from the help subtable for the parent window.

The HM_SET_ACTIVE_WINDOW message should be sent by ALL windows in response to the WM_ACTIVATE and WM_INITMENU messages as shown in the following example.

```

switch( usMsg )
{
    .
    .
    .

case WM_ACTIVATE:
    if( SHORT1FROMMP( mpl ) )
    {
        /*
         * Set active help window to this window's parent when
         * activated
         */
        WinSendMsg( WinQueryHelpInstance( hWnd ),
                    HM_SET_ACTIVE_WINDOW,

```

```

        WinQueryWindow( hWnd, QW_PARENT ),
        WinQueryWindow( hWnd, QW_PARENT ) );
    }
    else
    {
        /*
         * Clear active help window when this window is
         * deactivated - necessary for message box help, etc.
         * to work properly.
         */
        WinSendMsg( WinQueryHelpInstance( hWnd ),
                    HM_SET_ACTIVE_WINDOW,
                    NULL,
                    NULL );
    }
    break;

case WM_INITMENU:
    /*
     * Set active window to this window's parent here so that
     * the menu id will be found in the proper subtable.
     * Activation and deactivation of the help window will
     * take care of setting the help window back to the
     * active window.
     */
    WinSendMsg( WinQueryHelpInstance( hWnd ),
                HM_SET_ACTIVE_WINDOW,
                WinQueryWindow( hWnd, QW_PARENT ),
                WinQueryWindow( hWnd, QW_PARENT ) );
    break;

        .
        .
        .
}

```

When No Help Is Available

A user may request help by pressing F1 when the cursor is positioned on an item for which no field-level help is available. In such a case, IPF sends the HM_HELPSUBITEM_NOT_FOUND message to the application. To display the extended help window, the application then can either return FALSE or ignore the message. If the application returns TRUE, there is no response to the user request.

Help Window Resources

You can define the following window resources for the help interface:

- Help pull-down
- Help push button
- Command entry field
- Customized menu bar.

Help Pull-Down

CUA guidelines recommend that all application windows with menu bars include a help pull-down menu. The help application menu bar choice and corresponding pull-down menu is defined in your resource file. The following example shows how to define the help pulldown.

```

MENU IDR_MAIN PRELOAD
BEGIN
    SUBMENU "~File",
    BEGIN
        MENUITEM "~Open...",
        IDM_LOAD
    END

    SUBMENU "~Options",
    BEGIN
        SUBMENU "Size",
        BEGIN
            MENUITEM "Small",
            IDM_SIZE_SMALL, 0, MIS_TEXT
            MENUITEM "Medium",
            IDM_SIZE_MEDIUM, 0, MIS_TEXT
            MENUITEM "Large",
            IDM_SIZE_LARGE, 0, MIS_TEXT
            MENUITEM "Full Size",
            IDM_SIZE_FULL, 0, MIA_CHECKED
        END

        MENUITEM "~Jumble!",
        IDM_JUMBLE
    END

    SUBMENU "~Help",
    BEGIN
        MENUITEM "Help ~index",
        IDM_HELPINDEX, MIS_TEXT
        MENUITEM "~General help",
        IDM_HELPEXTENDED, MIS_TEXT
        MENUITEM "~Using help",
        IDM_HELPHELPFORHELP, MIS_TEXT
        MENUITEM SEPARATOR
        MENUITEM "~Product information",
        IDM_HELPABOUT, MIS_TEXT
    END
END

```

Help Push Button

If your application has a dialog or message area, you may want to include the Help push button in the bottom area of the secondary application window (dialog box). To define the Help push button, use the Presentation Manager button style BS_HELP and BS_NOPOINTERFOCUS.

The BS_HELP style causes the Presentation Manager to call IPF when the user selects this Help push button. The BS_NOPOINTERFOCUS style enables the Presentation Manager to determine the field for which the user requested help.

Command Entry Field

An entry field is a control window that enables users to enter text. A command entry field is used for typing commands, and may be programmed to accept entries recognized by the application.

For example, a command entry field might be used in an interpreter with a Presentation Manager interface. The field would accept a request from the user and execute it. Similarly, a command entry field might be used in an editor in a "command mode" to accept advanced instructions not associated with any editing keys. Any time the user has a limited number of correct responses, a command entry field may be appropriate.

Help windows for application commands can be associated with a command entry field by imbedding the index command tag (:icmd.) with a command name in the window that describes the command.

When the cursor is positioned in the associated entry field and the user presses F1 or selects the **Help** push button, titles of windows that contain these tags are displayed in alphabetic order in a list box window.

A Customized Menu Bar

A Help menu bar template is shipped with the *IBM Developer*. The template is in the HMTAILOR.RC file. Included in the template is the Help menu pull-down. You can customize the menu bar by adding pull-downs and choices to the Help menu bar template.

When a menu bar or pull-down choice you have added is selected by the application user, IPF sends the HM_ACTIONBAR_COMMAND

message to the currently active application window. The low-order word of *param1* contains the command value of the selected item. The command values of the actions added by the application must be between hex 7F00 and hex 7FFF for its commands.

The accelerator table maps function keys to commands on help windows. This table is also contained in the HMTAILOR.RC file. If you add a choice to the menu bar that maps to a key on the keyboard, you must also add an entry to the accelerator table for that choice. IPF functions depend on the entries that already exist in the system accelerator table. They must not be altered. The command value specified in the accelerator table entry must be the same command value that was specified for the associated action in the menu bar template.

If the HMTAILOR.RC file is changed, you must compile it using the resource compiler and attach it to the executable file. If the executable file is a DLL, you must call DosLoadModule to load it before calling WinCreateHelpInstance. Identify the handle to IPF in the hmodAccelActionBarModule field in the initialization structure. When this field is 0, IPF uses the default menu bar.

The HMTAILOR.RC file includes the HMTAILOR.H file.

Note: When modifying the menu bar, define IDM_HM_MENU and IDD_COVERPAGE_HM_ACCEL in your help header (.H) file. Also, add the IDs in the idActionBar and idAccelTable fields in the HELPINIT structure.

Customizing IPF with Communication Objects

The IPF online help interface and specific help information can be customized by hooking a piece of Presentation Manager code into the IPF help facility. This code, which is actually an entry point in a DLL file, is called a *communication object*.

Useful Communication Object Terminology

Before you begin reading about how to create IPF communication objects and related topics (such as application-controlled windows), you need to be familiar with the terms which will be used to describe them.

acviewport

The **:acviewport.** tag, which enables an application to control what is displayed in an IPF window (see [:acviewport. \(Application-Controlled Window\)](#)).

application-controlled window

The window that you create using a communication object that is loaded by an **:acviewport.** tag.

communication object

A child window of the object window; this child window can handle and respond to messages (particularly HM_ messages).

DLL

A dynamic link library (DLL) is a collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a DLL can be shared by several applications simultaneously.

entry point

A function called in the DLL that contains your communication object. Specified in the **:acviewport.** tag with the **objectname** option.

object

For the purposes of this discussion, synonymous with *entry point*.

window procedure

Code that is activated in response to a message. The window procedure controls the behavior and appearance of its associated windows.

Two Kinds of Communication Objects

How the communication object is loaded into memory is an indication of the purpose of the communication object. A DLL containing a communication object can be loaded into memory in two ways:

1. Through use of the **:docprof.** tag's **dll** attribute (see [:docprof. \(Document Profile\)](#)). A communication object loaded through the **:docprof.** tag is considered a "global" communication object because it affects the IPF coverage window and all of its children IPF windows. Essentially, a **:docprof.**-loaded communication object can alter the overall behavior and appearance of IPF, such as providing author-defined buttons in the control area or special message handling. You can load only one communication object in this manner.
2. Through use of the **:acviewport.** tags (see [:acviewport. \(Application-Controlled Window\)](#)). A communication object loaded in this way will only affect the specific application-controlled window and only when that window is open. Therefore, a communication object loaded through an **:acviewport.** tag is useful for tasks such as displaying video clips or bit maps associated with a given help panel. However, if you want a communication object to affect the overall appearance or behavior of the IPF interface for a given help session, you will want to use a "global" communication object (one that is loaded through the **:docprof.** tag). Unlike a global communication object, you can load multiple **:acviewport.** communication objects or refer to the same communication object from multiple **:acviewport.** tags.

An acviewport communication object is called each time a window is opened. A global communication object is called only once.

Purpose of Communication Objects

Communication objects can provide several powerful capabilities:

- You can write PM code that handles and responds to help-related messages. For this, you would use a **:docprof.**-loaded "global" communication object.
- You can write PM code to override the default appearance of the IPF help interface. For example, you can provide custom push buttons in the control area. For this, you would use a **:docprof.**-loaded "global" communication object.
- You can write PM code that allows help information to include function simulation, user interaction, and audio and video presentations (see [Controlling Windows with Applications \(ACVIEWPORTS\)](#)). For this, you would use an **:acviewport.**-loaded communication object.
- You can write PM code that changes help information (including .INF files) dynamically, with help window contents generated and formatted at the time when the information is displayed to the user. For this, you would use a **:docprof.**-loaded or "global" communication object. For more information about changing help information dynamically, see [Changing Help Information at Run Time \(DDF\)](#).

Essentially, a communication object is intended to provide message processing when no application is present to handle this message processing. In other words, your stand-alone PM applications can provide the same functionality that is supported by communication objects.

However, unlike a stand-alone application, a communication object that properly handles message passing can be used by any stand-alone application and can interact with other communication objects. For example, a communication object that provides system call support (such as querying specific details about the user's system) might be useful to a broad range of applications in their online information.

Understanding Communication Objects

Before you begin coding, you should understand the general steps involved in the creation and function of a communication object. The steps are as follows:

1. IPF encounters a **:docprof.** or **:acviewport.** tag.
2. Based on the content of the tag, IPF loads a DLL and calls a function that corresponds to a communication object.
3. IPF passes 2 items to the communication object:
 - a. A pointer to an ACVP data structure. This data structure includes the following elements:

```

ULONG  cb;           /* length of data structure          */
HAB     hAB;         /* anchor block handle              */
HMq     hmQ;         /* handle to message queue          */
ULONG   ObjectID;    /* ObjectID attribute as specified in */
                /* an acviewport tag                */
HWND    hWndParent;  /* handle to acviewport parent window */
HWND    hWndOwner;   /* handle to acviewport owner window  */
HWND    hWndACVP;    /* handle to acviewport              */

```

IPF supplies all but the last piece of information in this data structure for the communication object. If the communication object creates an application-controlled window, it must place the handle to that window in the last element of the data structure before returning to IPF. IPF uses the handle to size and position the window.

If the communication object is global (loaded by means of a **:docprof.** tag), the last 3 items are not needed and are set to null by IPF.

- b. A string identified in the **:docprof.** or **:acviewport.** tag as **objectinfo**. This is simply a generic parameter passed to the communication object. Your communication object may or may not use this parameter, but it should process it.
4. The communication object begins execution. It must complete the following three steps, sequentially:
 - a. Register a window class.
 - b. Create a window.
 - c. Insert the window in the communication chain (see [The Communication Chain](#)).
5. The next step depends on whether or not the communication object DLL was loaded through a **:docprof.** tag or an **:acviewport.** tag.
 - If the communication object DLL was loaded through a **:docprof.** tag, it simply returns.
 - If the communication object DLL was loaded through an **:acviewport.** tag, the communication object passes the window handle of the window it created to IPF.
6. The communication object begins message processing:
 - If the communication object DLL was loaded through **:docprof.** tag, it must process HM_ messages and the WM_CLOSE message.
 - If the communication object DLL was loaded through an **:acviewport.** tag, it must process:
 - HM_ messages
 - Various WM_ messages
 - Optionally, other messages, dependent on the particular design of the communication object.

Messages between IPF and the Communication Object

IPF and its communication objects communicate through window and help manager messages. IPF communication objects, and windows that they create, can send messages to IPF and IPF windows for which the communication objects can get a handle. Similarly, IPF can send messages to any window that a communication object creates.

Any message that an application can send to IPF also can be sent by IPF communication objects. As a programmer working with communication objects, you will need to understand the following important help manager messages:

- HM_CONTROL
- HM_INFORM
- HM_INVALIDATE_DDF_DATA
- HM_NOTIFY
- HM_QUERY
- HM_QUERY_DDF_DATA
- HM_SET_COVERPAGE_SIZE
- HM_SET_OBJCOM_WINDOW
- HM_SET_USERDATA
- HM_UPDATE_OBJCOM_WINDOW_CHAIN

For a detailed description of these messages and other IPF messages, see [Help Manager Messages](#).

The Communication Chain

Communication objects, whether they are loaded by means of a **:docprof.** tag or an **:acviewport.** tag, function in a "communication chain." The chain is maintained through the passing of HM_ messages unidirectionally through the chain.

IPF passes all messages to the active communication object (including, but not limited to, HM_ messages) as they are generated by the user's actions. The active communication object is then responsible for passing on any HM_ messages it receives through the chain to a neighboring communication object. The handle of the neighboring communication object window is returned by the HM_SET_OBJCOM_WINDOW (see [Adding Your Communication Object to the Communication Chain](#)).

Even if you are certain that your communication object will not be used in combination with other communication objects, it is your responsibility to code your communication object contingent on the possibility that other communication objects may be present in the chain. Therefore, it is important for you to create communication objects that keep the communication chain intact through message passing.

Making Your Communication Objects Function in the Chain

As a programmer creating communication objects, you need to be certain that your communication object can successfully complete the following communication chain tasks:

1. Your communication object must add itself to the communication chain.
2. If a communication object that was added to the chain before your communication object was added removes itself from the chain, your communication object must respond to keep the chain intact.
3. Your communication object must be able to remove itself from the chain and keep the chain intact.

Adding Your Communication Object to the Communication Chain

Even if your communication object is the only one in use, your communication object is still part of the communication chain. To add itself to the chain, your communication object must:

1. Send the help manager message HM_SET_OBJCOM_WINDOW to the window *PACVP->hWnd Parent* .
Your communication object must pass its frame handle as the first parameter.
2. Your communication object must receive and **store** the window handle it receives in return.
3. Your communication object should pass all HM_ help manager messages to that window handle.

The following code fragment illustrates this process.

```
#define INCL_WIN
#define INCL_WINHELP
#include <os2.h>

#define HM_MSG_MAX (HM_MSG_BASE+0x0024)

USHORT  IPFClassRegistered = 0;          /* IPF class registered flag      */

/* Main Entry point */
MRESULT EXPENTRY IPFMain (PACVP pACVP, PCH Parameter);

/* Pop up error box */
VOID Error (PCH str);

MRESULT EXPENTRY IPFMain (PACVP pACVP, PCH Parameter)
/* pACVP contains the following structure:

typedef struct_ACVP
{
    ULONG  cb;                length
    HAB    hAB;               anchor block handle
    HMQ     hmq;               messge queue handle
    ULONG  ObjectID;          object identifier
    HWND    hWndParent;       IPF viewport client handle
    HWND    hWndOwner;        IPF viewport client handle
    HWND    hWndACVP;         applications frame window hwnd
} ACVP, *PACVP;
```

This structure is prefilled in except for hWndACVP. You must put your window handle there if you are creating an application-controlled viewport. If this is just a generic communication object, you do not need to fill it in.

Parameter is the information passed in with the objectinfo tag.

Note: You can use ObjectID to have multiple acviewports and comm objects use the same entry point. Check the ObjectID to find out where you were called from in the IPF file. */

```
{
    HWND  hwndFrame, hwndPrevious, hwndLatest, hwndClient;
    ULONG CtrlData = 0;

    Error ( Parameter );
    /* Check global to see if our window class has been registered. */
    /* if not, register it. */
    if (!IPFClassRegistered)
    {
        /* We will register our class with 4 extra bytes of information
           so that we can place the previous object comm window handle there.
           You might want to create a structure store here instead. */
        if (!WinRegisterClass( pACVP->hAB,
                               "CLASS_IPF",
                               (PFNWND) IPF_WinProc,
                               CS_SYNCPAINT | CS_SIZEREDRAW | CS_MOVENOTIFY,
                               4))
        {
            Error ("Can not register class");
            exit (TRUE);
        }
        IPFClassRegistered = 1;
    }

    /* Create window.  Visibility does not matter, as IPF will take care
       of it. */
    if (!(hwndFrame = WinCreateStdWindow (pACVP->hWndParent,
                                          WS_VISIBLE,
                                          &CtrlData,
                                          "CLASS_IPF",
                                          "IPF",
                                          0L,
                                          0L,
                                          0L,
                                          &hwndClient
                                          )))
    {
        Error ("Can not create window");
        return (MRESULT) TRUE;
    }

    /* Setup our window in the ACVP structure.  This is only necessary if
       you are creating an acviewport. */
    pACVP->hWndACVP = hwndFrame;

    /* Set the current comm object window to us */
    hwndPrevious = (HWND) WinSendMsg (pACVP->hWndParent,
                                      HM_SET_OBJCOM_WINDOW,
                                      (MPARAM) hwndFrame,
                                      (MPARAM) NULL);

    /* Query back the comm obj window */
    hwndLatest = (HWND) WinSendMsg (pACVP->hWndParent,
                                    HM_QUERY,
                                    MPFROM2SHORT ((USHORT) 0, HMQW_OBJCOM_WINDOW),
                                    (MPARAM) NULL);

    /* double check to make sure we are in the comm chain */
    if (hwndFrame != hwndLatest)
    {
        Error ("Can not set object communication window");
        return (MRESULT) TRUE;
    }

    /* Store the previous commobj handle in window words */
    if (!WinSetWindowUlong (hwndClient, QWL_USER, (ULONG) hwndPrevious))
    {
        Error ("Can not save handle into reserved memory");
        return (MRESULT) TRUE;
    }
    return (MRESULT) FALSE;
}
```



```

VOID Error (PCH str)
{
    WinMessageBox (HWND_DESKTOP,
                  HWND_DESKTOP,
                  (PCH)str,
                  (PCH)"IPF Sample Error Message",
                  1,
                  MB_OK | MB_APPLMODAL |
                  MB_MOVEABLE | MB_ICONASTERISK);
}

```

Responding to the Removal of Another Object in the Chain

If a communication object that was added to the chain before your communication object removes itself from the chain, your communication object must respond to keep the chain intact.

This involves several steps:

1. Your communication object must process the HM_UPDATE_OBJCOM_WINDOW_CHAIN help manager message. This message provides your communication object with:
 - a. The handle of the communication object that is removing itself from the chain.
 - b. The window handle for the object to serve as a replacement in the chain.

Your communication object does not need to respond unless the first parameter, the handle of the communication object removing itself, is identical to the handle returned to your communication object when it initially sent a HM_SET_OBJCOM_WINDOW message to add itself to the chain (see [Adding Your Communication Object to the Communication Chain](#)). Your communication object should have stored this handle.

2. If the second parameter returned by HM_UPDATE_OBJCOM_WINDOW_CHAIN does not match the handle previously returned by HM_SET_OBJCOM_WINDOW, your communication object is not affected by the modification to the chain and does not need to respond other than to pass on the message to the next communication object in the chain.

However, if these handles are identical, your communication object has a new "neighbor" in the chain, and should begin passing all HM_help manager messages to the object identified by the handle returned as the first parameter of HM_UPDATE_OBJCOM_WINDOW_CHAIN.

The following code fragment illustrates this process.

```

#define INCL_WIN
#define INCL_WINHELP
#include <os2.h>

#define HM_MSG_MAX (HM_MSG_BASE+0x0024)

USHORT  IPFClassRegistered = 0;          /* IPF class registered flag */

/* Window procedure */
MRESULT EXPENTRY IPFWinProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);

MRESULT EXPENTRY IPFWinProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
/* The window procedure will handle deleting ourselves from the chain
   as well as forwarding all standard IPF message on the the next window in
   the chain */
{
    HWND hwndPrevious, hwndLatest;

    /* Get previous comm obj */
    hwndPrevious = (HWND) WinQueryWindowULong (hwnd, QWL_USER);
    /* Handle the messages you want here, but do not return unless you
       now the message will not need to be handled by another comm obj.
       Let the the message be forwarded on to the next comm obj. The
       only exception to this is if you change the cover page size. You
       need to return TRUE to prevent the coverpage from being resized. */
    switch (msg)
    {
        case HM_UPDATE_OBJCOM_WINDOW_CHAIN:
            /* If another window is being inserted, replace previous with it */

```

```

        if (hwndPrevious == (HWND)mp1) {
            hwndPrevious = (HWND)mp2;
            if (!WinSetWindowULong (hwnd, QWL_USER, (ULONG) hwndPrevious)) {
                /* Put up error message */
                WinMessageBox (HWND_DESKTOP,
                               HWND_DESKTOP,
                               (PCH)"Can not save handle into reserved memory",
                               (PCH)"IPF Sample Error Message",
                               1,
                               MB_OK | MB_APPLMODAL |
                               MB_MOVEABLE | MB_ICONASTERISK);
            }
            break;
        }

    } else {
        /* Otherwise simply forward the message on */
        if (hwndPrevious != 0L) {
            WinSendMsg (hwndPrevious, HM_UPDATE_BJCOM_WINDOW_CHAIN,
                        (MPARAM) mp1, (MPARAM) mp2);
        }
    }
    return (MPARAM) NULL;
break;
case WM_CLOSE:
    WinDestroyWindow (WinQueryWindow (hwnd, QW_PARENT));
    return (MPARAM) NULL;
break;
case WM_DESTROY:
    /* Take ourselves out of the chain */
    hwndLatest = (HWND)WinSendMsg (hwnd, HM_QUERY,
                                   MPFROM2SHORT (0, HMQW_OBJCOM_WINDOW),
                                   (MPARAM) NULL);
    WinSendMsg (hwndLatest, HM_UPDATE_OBJCOM_WINDOW_CHAIN,
                (MPARAM) WinQueryWindow (hwnd, QW_PARENT),
                (MPARAM) hwndPrevious );
    return (MPARAM) NULL;
break;
}
if ((msg > HM_MSG_BASE) && (msg <= HM_MSG_MAX)) {
    return WinSendMsg (hwndPrevious, msg, mp1, mp2);
} else {
    return WinDefWindowProc (hwnd, msg, mp1, mp2);
} /* endif */
}

```

Removing Your Communication Object from the Chain

Your communication object must be able to remove itself from the chain and keep the chain intact. When the communication object does remove itself, it is no longer the active object. This can be carried out as follows:

1. Your communication object must process the WM_DESTROY message sent by IPF that corresponds to the closure of a window with which your communication object is associated.
2. Your communication object must query the current object window using HM_QUERY. This returns the handle of the current object window.
3. Your communication object can then send the HM_UPDATE_OBJCOM_WINDOW_CHAIN message to the handle of the current object window including the handle of your communication object, which was returned when you added your communication object to the chain (see [Adding Your Communication Object to the Communication Chain](#)).

The following code fragment illustrates this process:

```

case WM_DESTROY:

    hwndPrevious = (HWND) WinQueryWindowULong (hwnd, COM_HWND);

    hwndLatest = (HWND) WinSendMsg (hwnd,
                                    HM_QUERY,
                                    MPFROM2SHORT ((USHORT) 0, HMQW_OBJCOM_WINDOW),
                                    NULL);

```

```

WinMessageBox (HWND_DESKTOP,
              HWND_DESKTOP,
              (PCH)str,
              (PCH) "IPF Sample Error Message",
              1,
              MB_OK | MB_APPLMODAL |
              MB_MOVEABLE | MB_ICONASTERISK);
WinSendMsg (hwndLatest,
            HM_UPDATE_OBJCOM_WINDOW_CHAIN,
            (MPARAM) hwndPrevious,
            (MPARAM) WinQueryWindow (hwnd, QW_PARENT));

rValue = FALSE;
break;

default:

    rValue = TRUE;
    break;
}
}

```

Changing the Coverage Window

One of the IPF customizations you can do with your communication object is manipulating the IPF coverage window.

When an online book is opened or when an application requests that IPF create a help instance, IPF creates a *coverage* frame window. The coverage window is the window in which all other information is displayed. IPF-controlled windows are children of the coverage window, as are application-controlled windows.

IPF communication objects can change the look and functionality of the coverage window. For example, a communication object can change the size or location of the coverage, or remove its menu bar.

For HLP files, applications can change only the size of a help window. The application would do this by sending HM_SET_COVERAGE_SIZE to the help instance created with [WinCreateHelpInstance](#). Note that applications must not move the help window.

For INF files, the communication object must handle the OPEN_COVERAGE event of the HM_NOTIFY message, then use WinSetWindowPos to set the size and position of the coverage. The coverage window's handle is in *mp2*.

When using WinSetWindowPos to change the position and size of the coverage window, you must OR SWP_SHOW with SWP_MOVE or SWP_SIZE, or both.

You must return directly from your code when you handle this event. Do not pass the HM_NOTIFY message on to the default window procedure.

In addition to the techniques specified here for resizing and positioning the coverage, once you have the handle to the coverage you can use any of the other PM window manipulation techniques, such as manipulating the frame controls or menus. For example, to remove the coverage's menu bar, you would call WinDestroyWindow in response to an OPEN_COVERAGE event, as in the following example:

```
WinDestroyWindow (WinWindowFromID (HWND) mp2, FID_MENU);
```

Use other FID_* IDs to work with other parts of the coverage window.

The code fragment in the following figure illustrates these techniques by changing the size and location of the coverage and removing the menu bar.

```

HWND hwndCoverPage;
.
.
.
/* Inside the main switch statement in your window procedure... */
case HM_NOTIFY:

    switch (SHORT1FROMMP (mp1)) {

        case OPEN_COVERAGE:

```

```

hwndCoverPage = (HWND) mp2; /* Handle to coverpage is in mp2 */

/* Remove the menu */
WinDestroyWindow(WinWindowFromID(hwndCoverPage, FID_MENU));

/* Change the size and location of coverpage */
WinSetWindowPos(hwndCoverPage, 0, 10, 10, 200, 200,
    SWP_SIZE | SWP_MOVE | SWP_SHOW);

return TRUE;

} /* endswitch */

break;

```

Controlling Windows with Applications (ACVIEWPORTS)

IPF handles the formatting and display of text and graphic information within its windows. *IPF-controlled* windows are defined in the tagged source file with a heading tag or **:link..** These windows are IPF-controlled because IPF provides the window procedures that control them. The content and presentation of information in an IPF-controlled window is limited by the functions of a standard OS/2 window.

To create IPF-controlled windows, an author requires only tagging skills; however, to create *application-controlled* windows, an author requires both tagging and programming skills.

Application-controlled windows are defined in the tagged source file with the application-controlled window tag (**:acviewport.**). With this tag, a window is controlled by a communication object (see [Customizing IPF with Communication Objects](#)) that has been written and compiled into the form of a dynamic link library (DLL). When an IPF window is displayed at execution time and **:acviewport.** is encountered, IPF passes control to the entry point in the DLL specified by the **objectname=** attribute of **:acviewport.** This entry point is, in fact, a communication object.

At this point, the communication object takes control and executes the instructions specified in the source file. When the call returns to IPF, IPF sizes and positions the window on the screen as defined in the heading tag or **:link.** (see [Customizing Windows](#)).

The **:acviewport** tag can share a panel with text and, for that matter, you can have more than one **:acviewport** tag in the same panel.

In addition, an **acviewport** communication object can be called from more than one panel. The **objectid=** attribute enables the communication object to determine which help panel called it.

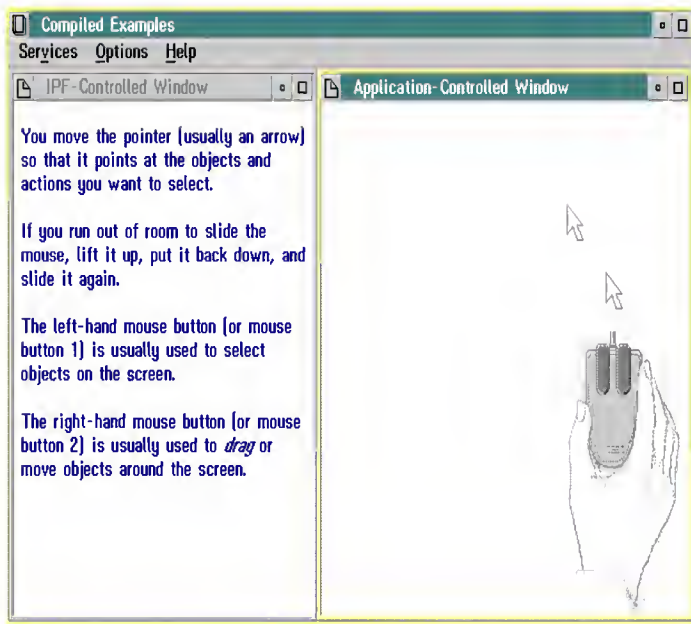
The following figure shows the tagging to produce a split window. In this example, the contents of the left window are IPF-controlled. The contents of the right window are defined and controlled by the IPF communication object **IPFMain** which resides in IPF.DLL.

```

:userdoc.
:title.Information Presentation Facility
:docprof ctrlarea=none.
:h1 res=016 scroll=none clear.Using a Mouse
.*
:link reftype=hd res=017 auto split
    vpx=left vpy=top vpcx=50% vpcy=100%
    rules=border scroll=none titlebar=none.
.*
:acviewport dll='ipf'
    objectname='IPFMain' objectid=1
    vpx=right vpy=top vpcx=50% vpcy=100%.
.*
:h2 res=017.Using a Mouse
.*
:p.You move the pointer (usually an arrow) so that it
points at the objects and actions you want to select.
:p.If you run out of room to slide the mouse, lift it up, put it
back down, and slide it again.
:p.The left-hand mouse button (or mouse button 1) is
usually used to select objects on the screen.
:p.The right-hand mouse button (or mouse button 2) is
usually used to :hp1.drag:ehp1. or move
objects around the screen.
:userdoc.

```

The following figure shows the compiled version of the tagging shown in the previous figure.



The left window is IPF-controlled. The right window displays an animated mouse whose activity is controlled by a routine in a DLL.

In the previous example, IPF processes **:acviewport**. as follows:

1. It loads IPF.DLL and calls the procedure IPFMain. This object name is case sensitive.
2. IPFMain creates a window and registers it with IPF as an object communication window.
3. When the call to IPFMain returns to IPF, IPF gives instructions to display the animated mouse.

IPFMain, IPF.DLL, and the bit maps used for the animated mouse are provided in the IPF sample program available with the Toolkit.

Writing the Communication Object Code

An IPF communication object can be structured in many ways. Its content depends on the function being implemented. Application-controlled windows typically simulate activity that might or might not require user interaction.

An example of a communication object is provided in the IPF sample program (available in the Toolkit), and is shown in the previous IPF-controlled window example. The program contains two procedures:

IPFMain registers a window class for the application-controlled window, creates an instance of the class, and registers it with IPF as a communication object.

IPFWinProc provides the animation in the application-controlled window. IPFWinProc is called by IPFMain procedure.

```
#define INCL_WIN
#define INCL_GPI
#define INCL_DOS
#define INCL_DOSMODULEMGR
#define LINT_ARGS
#define DINCL_32

#include <OS2.H>
#include "IPF.H"

#define COM_HWND          4 /* Used in WinSetWindowULong */
#define FRAMES            5 /* Number of frames in animation sequence */
#define BEEP_WARN_FREQ    60 /* Frequency of warning beep */
#define BEEP_WARN_DUR     100 /* Duration of warning beep */

USHORT  IPFClassRegistered = 0; /* IPF class registered flag */
```

```

HWND      hwndClient;          /* Handle to the client window */
HWND      hwndPrevious;        /* Handle to the previous active */
/* object communication window */
HWND      hwndLatest;          /* Handle to the latest active */
/* object communication window */

MRESULT EXPENTRY IPFMain (PACVP pACVP, PCH Parameter);
MRESULT EXPENTRY IPFWinProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);
VOID Error (PCH str);

MRESULT EXPENTRY IPFMain (PACVP pACVP, PCH Parameter)
{
    HWND hwndParent;           /* Handle of parent window in IPF */
    HWND hwndFrame;            /* Handle to the frame */
    ULONG WinStyle;             /* window style for creating frame */
    ULONG CtrlData;            /* control data for creating frame */

    Parameter; /* Warning Level 3 Avoidance */

    /** 1) Initialize **/
    if (!IPFClassRegistered)
    {
        if (!WinRegisterClass (pACVP->hAB,
                                "CLASS_IPF",
                                (PFNWP) IPFWinProc,
                                CS_SYNCPAINT | CS_SIZEREDRAW | CS_MOVENOTIFY,
                                8))
        {
            DosBeep (BEEP_WARN_FREQ, BEEP_WARN_DUR);
            exit (TRUE);
        }
        IPFClassRegistered = 1;
    }
    WinStyle = 0L;
    CtrlData = 0L;

    if (!(hwndFrame = WinCreateStdWindow (pACVP->hWndParent,
                                           WinStyle,
                                           &CtrlData,
                                           "CLASS_IPF",
                                           "IPF",
                                           0L,
                                           0L,
                                           0L,
                                           &hwndClient
                                           )))
    {
        Error ("Cannot create window");
        return (MRESULT) TRUE;
    }

    /** 2) Process **/

    pACVP->hWndACVP = hwndFrame;

    hwndParent = pACVP->hWndParent;

    hwndPrevious = WinSendMsg (pACVP->hWndParent,
                               HM_SET_OBJCOM_WINDOW,
                               (MPARAM) hwndFrame,
                               NULL);

    hwndLatest = WinSendMsg (pACVP->hWndParent,
                             HM_QUERY,
                             MPFROM2SHORT (NULL, HMQW_OBJCOM_WINDOW),
                             NULL);

    if (hwndFrame != hwndLatest)
    {
        Error ("Cannot set object communication window");
        return (MRESULT) TRUE;
    }

    /** 3) Finish **/

    if (!WinSetWindowULong (hwndClient, COM_HWND, (ULONG) hwndPrevious))
    {
        Error ("Cannot save handle into reserved memory");
        return (MRESULT) TRUE;
    }

```

```

    }
    return (MRESULT) FALSE;
}

```

```

MRESULT EXPENTRY IPFWinProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{

```

```

    static HAB      Hhab;          /* anchor block handle          */
    static HBITMAP  hbm [5];       /* array of bitmap handles     */
    static HPS      hps;          /* presentation space          */
    static POINTL   ptl;          /* pointl                      */
    static HMODULE   hModule;      /* to get bitmaps from DLL resource */
    static SHORT    index;        /* index to current bitmap to display */
    static LONG      cxClient,
                    cyClient;     /* window size                 */
    BOOL            rValue=TRUE;   /* FALSE if the message was acted */
                                /* upon successfully           */

```

```

/** 1) Initialize */

```

```

switch (msg)
{
    case HM_UPDATE_OBJCOM_WINDOW_CHAIN:

        hwndPrevious = (HWND) WinQueryWindowULong (hwnd, COM_HWND);

        if (hwndPrevious == mp2)
        {
            hwndPrevious = mp1;

            if (!WinSetWindowULong (hwndClient,
                                    COM_HWND,
                                    (ULONG) hwndPrevious))
            {
                Error ("Cannot save handle into reserved memory");
                break;
            }
        }
        else
        {
            if (hwndPrevious != NULL)
            {
                WinSendMsg (hwndPrevious,
                            HM_UPDATE_OBJCOM_WINDOW_CHAIN,
                            (MPARAM) mp1,
                            (MPARAM) mp2);
            }
        }

        rValue = FALSE;
        break;

```

```

    case WM_CREATE:

```

```

        if (DosLoadModule (NULL, 0L, "IPF", &hModule))
        {
            Error ("Cannot load module");
            break;
        }

        if (!(hps = WinGetPS(hwnd)))
        {
            Error ("Cannot get presentation space");
            break;
        }

        for (index = 0; index < FRAMES; index++)
        {
            if (!(hbm [index] = GpiLoadBitmap (hps,
                                                hModule,
                                                (USHORT) (IDB_FRAME1+index),
                                                cxClient,
                                                cyClient)))
            {
                Error ("Cannot load bitmap");
                return (MRESULT) rValue;
            }
        }

```

```

    }

    WinReleasePS (hps);

    index = 0;

    if (! (Hhab = WinQueryAnchorBlock (hwnd)))
    {
        Error ("Cannot retrieve anchor block handle");
        break;
    }

    if (!WinStartTimer (Hhab, hwnd, ID_TIMER, 150))
    {
        Error ("Cannot start timer");
        break;
    }

    rValue = FALSE;
    break;

case WM_TIMER:

    if (index++ == FRAMES-1)
    {
        index = 0;
    }

    WinInvalidateRect (hwnd, NULL, FALSE);

    rValue = FALSE;
    break;

/** 2) Process **/

case WM_PAINT:

    if (! (hps = WinBeginPaint (hwnd, NULL, NULL)))
    {
        Error ("Cannot set presentation space for drawing");
        break;
    }

    if (!WinDrawBitmap (hps,
                        hbm [index],
                        NULL,
                        &ptl,
                        CLR_NEUTRAL,
                        CLR_BACKGROUND,
                        DBM_NORMAL))
    {
        Error ("Cannot draw bitmap");
        break;
    }

    WinEndPaint (hps);

    rValue = FALSE;
    break;

case WM_SIZE:

    cxClient = SHORT1FROMMP (mp2);
    cyClient = SHORT2FROMMP (mp2);

    rValue = FALSE;
    break;

/** 3) Finish **/

case WM_CLOSE:

    WinDestroyWindow (WinQueryWindow (hwnd, QW_PARENT));

    rValue = FALSE;
    break;

case WM_DESTROY:

```



```

WinStopTimer (Hhab, hwnd, ID_TIMER);

for (index = 0; index < 8; index++)
{
    GpiDeleteBitmap (hbm [index]);
}

hwndPrevious = (HWND) WinQueryWindowULong (hwnd, COM_HWND);

hwndLatest = WinSendMsg (hwnd,
                        HM_QUERY,
                        MPFROM2SHORT (NULL, HMQW_OBJCOM_WINDOW),
                        NULL);

WinSendMsg (hwndLatest,
            HM_UPDATE_OBJCOM_WINDOW_CHAIN,
            (MPARAM) hwndPrevious,
            (MPARAM) WinQueryWindow (hwnd, QW_PARENT));

DosFreeModule (hModule);

rValue = FALSE;
break;

default:

    rValue = TRUE;
    break;

}

return (rValue ? WinDefWindowProc (hwnd, msg, mp1, mp2) : 0L);
}

```

Using Communication Windows

To position windows or graphics within an application-controlled window, the IPF communication object requires a communication object window. For example, an application-controlled window can be used to represent the workplace, with an interactive, simulated application window positioned on the workplace. However, because IPF sizes and positions the application-controlled window after returning from the call to a communication object, the communication object cannot size and position the simulated application window until after it has created the window and returned control to IPF.

The dilemma is resolved because the communication object can receive HM_INFORM messages after **:acviewport.** has been processed by IPF and the communication object has created an active communication object window. Upon receiving the HM_INFORM message from IPF, the window procedure can then create the simulated application window and position it within the application-controlled window.

The following C-language source code contains the communication object ComWindow that creates a communication window and processes messages from IPF.

```

#define INCL_WIN
#define INCL_DOS

#include <os2.h>

/* Define ID used with reftype = inform attribute in the link tag */
/* in tagged source for help information */

#define SIMULATE_APPWINDOW 1000

MRESULT EXPENTRY ComWindowProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);
MRESULT EXPENTRY SimWindowProc (HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2);

HWND hComWindow = NULL;
HWND hSimWindow = NULL;
HWND hComClientWindow;
HWND hSimClientWindow;
HWND PreviousComWindow;
HWND PreviousHwnd;

USHORT EXPENTRY ComWindow (pACVP, ObjectInfo)

```



```

        WinSetWindowText (hSimWindow, "Application X");

        WinSendMsg (hSimWindow,
                    WM_SETICON,
                    WinQuerySysPointer (HWND_DESKTOP, SPTR_APPICON,
                    FALSE), NULL);

        /* get the size of the communication client window */

        WinQueryWindowRect (hwnd, &Rect);

        /* adjust the size of the application window within the */
        /* communication client window */

        Rect.xLeft = Rect.xRight / 12;
        Rect.yBottom = Rect.yTop / 5;
        Rect.xRight = Rect.xLeft * 10;
        Rect.yTop = Rect.yBottom * 3;

        /* position the application window within the */
        /* communication client window */

        WinSetWindowPos (hSimWindow, HWND_TOP,
                        (SHORT)Rect.xLeft,
                        (SHORT)Rect.yBottom,
                        (SHORT)Rect.xRight,
                        (SHORT)Rect.yTop,
                        (SWP_SHOW | SWP_SIZE |
                        SWP_MOVE | SWP_ACTIVATE));

        return (MRESULT)TRUE;
    }

    case WM_PAINT:

        hps = WinBeginPaint (hwnd, (HPS)NULL, (PRECTL)NULL);
        WinQueryWindowRect (hwnd, &Rect);
        WinFillRect (hps, &Rect, CLR_RED);
        WinEndPaint (hps);
        break;

    case WM_CLOSE:

        WinDestroyWindow (WinQueryWindow (hwnd, QW_PARENT));
        return (MRESULT)TRUE;

    case WM_DESTROY:

        PreviousHwnd = (HWND)WinQueryWindowULong (hwnd, 0L);
        WinSendMsg (WinQueryWindow (hwnd, QW_PARENT),
                    HM_SET_OBJCOM_WINDOW,
                    PreviousHwnd,
                    NULL);

        break;
    }

    return (WinDefWindowProc (hwnd, msg, mp1, mp2));
}

/* Create the simulated frame window */

MRESULT EXPENTRY SimWindowProc(HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
    HPS        hps;
    RECTL      Rect;

    switch (msg)
    {
        case WM_PAINT:

            hps = WinBeginPaint (hwnd, (HPS)NULL, (PRECTL)NULL);
            WinQueryWindowRect (hwnd, &Rect);
            WinFillRect (hps, &Rect, CLR_WHITE);
            WinEndPaint (hps);
            break;
    }

```

```

        case WM_CLOSE:

            WinDestroyWindow (WinQueryWindow (hwnd, QW_PARENT));
            return (MRESULT)TRUE;

        case WM_DESTROY:

            PreviousHwnd = (HWND)WinQueryWindowULong (hwnd, 0L);
            WinSendMsg (WinQueryWindow (hwnd, QW_PARENT),
                        HM_SET_OBJC_WINDOW,
                        PreviousHwnd,
                        NULL);

            break;

    }

    return (WinDefWindowProc (hwnd, msg, mp1, mp2));
}

```

The following shows the tagging that communicates with the communication object through the **reftype=inform** attribute of **:link**.. The contents of the right window are defined by IPF. The contents of the left window are defined and controlled by the communication object **ComWindow** and resides in **INFORM.DLL**.

```

:userdoc.
:docprof ctrlarea=none.
:h1 id=examp5 scroll=none
    x=left y=bottom width=100% height=100%.
Interacting with Application Windows on the Workplace
.*
:link reftype=hd refid=mytxt5
    vpx=left vpy=bottom vpcx=50% vpcy=100%
    titlebar=none scroll=none auto split.
.*
:acviewport dll='inform'
    objectname='ComWindow'
    objectid=1
    objectinfo='optional'
    vpx=right vpy=bottom
    vpcx=50% vpcy=100%.
.*
.*
:link reftype=inform res=1000 auto.
:h1 hide id=mytxt5.My text
:p.
This window could contain an explanation of how to interact with
the application-controlled window displayed on the right.
:userdoc.

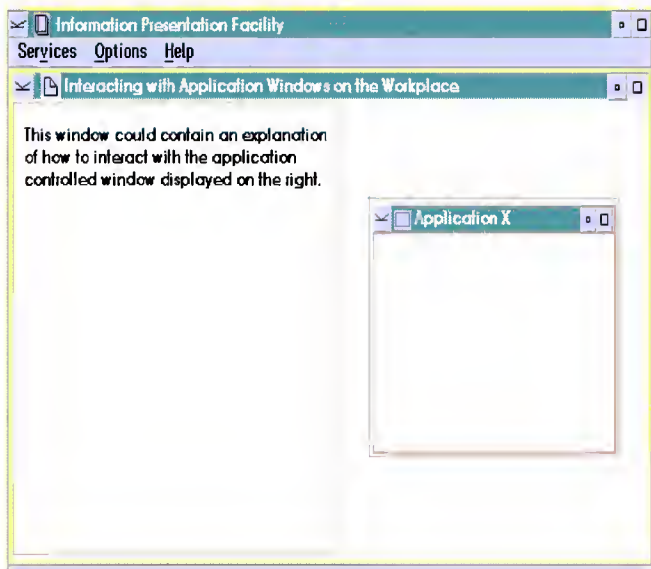
```

In the previous example, IPF processes **:acviewport**. as follows:

1. It loads **INFORM.DLL** and calls the procedure **ComWindow**.
2. **ComWindow** passes the value of **objectid=** and **objectinfo=**. These attributes are place holders for this example.
3. **ComWindow** creates a communication window that will receive the **HM_INFORM** messages from IPF when it processes the **reftype=inform** attribute of **:link**..

When the **HM_INFORM** message is sent to IPF, IPF creates and displays the simulated application window.

The following figure displays the windows from the previous tagging example.



An Application-Controlled Window. The communication object window is a functioning frame window.

Communication windows also are useful when the same communication object is used to support multiple application-controlled windows in help information. For example, you can use the same IPF communication object to represent different simulated application windows from one window to another. Using the previous examples, this is accomplished in two steps.

- Add another **:h1.** window definition to the tagged source for the help information. A different number is specified in the **res=** attribute for the **:link.** tag that has the **reftype=inform** attribute
- Add the corresponding **res=** number as another possible value of the parameter to the HM_INFORM message. It is processed accordingly by the communication object window procedure

Changing Help Information at Run Time (DDF)

Dynamic data formatting (DDF) allows you to incorporate text, bit maps, or metafiles in an IPF window at execution time. You can use the dynamic data formatting facility in conjunction with the dynamic data format tag (**:ddf.**). The **:ddf.** tag functions as a request by IPF to the application for the DDF data, and a set of DDF application programming interface calls that provide primitives for formatting text. The DDF calls also allow you to incorporate bit maps and metafiles dynamically, and to specify a hypertext or inform link from DDF data to non-DDF data.

IPF has no knowledge of the DDF data it displays, other than that a block of data has been provided to it by the application program. Therefore, DDF data cannot be searched or printed. In effect, DDF is a specific extension of application-controlled windows. When the **:ddf.** tag is encountered at execution time, IPF sends the HM_QUERY_DDF_DATA message to the application window procedure with which the current instance of help is associated. IPF sends the message either by a WinAssociateHelpInstance request or a HM_SET_OBJCOM_WINDOW message.

DDF and Online Help

DDF data is treated differently for a help and an online document. In the case of a help facility, the HM_QUERY_DDF_DATA message must be processed in the application's window procedure. Within the processing for this message, you can turn on the number specified in the **res=** attribute of the **:ddf.** tag to allow for different processing based on which IPF window with a **:ddf.** tag is currently being displayed.

Therefore, in the case of dynamic data formatting within help, it is not necessary to specify an application-controlled window or a separate DLL. However, this would also work if the application-controlled window used the HM_SET_OBJCOM_WINDOW message to explicitly identify the entry point specified in the **dll=** ' ' and **objectname=** ' ' attributes of the **:acviewport.** tag as the proper window procedure where the HM_QUERY_DDF_DATA message is processed.

DDF and Online Documents

The situation is different with an online document (as in, filetype INF). To display DDF data in an online document, the **:ddf** tag must be specified within an application-controlled window. The window that actually specifies the **:ddf** tag must be defined as a LINK AUTO SPLIT of the application-controlled window's parent window that is specified with a heading tag. The reason is based on the serialization sequence when IPF reads an .INF source file. For example, suppose the file is tagged as follows:

```
:h1 res=100 x=0 y=0 width=50% height=50%.DDF Parent
:acviewport dll='test.dll' objectname='someobject' objectid='1'.
:ddf res=100.
```

The HM_QUERY_DDF_DATA message will be sent to the window procedure of VIEW, which does not process it, and it will be lost. However, suppose the tagging sequence is as follows:

```
:h1 res=100 x=left y=top width=100% height=100% titlebar=both clear.Look here first
:acviewport dll='flight' objectname='GetName' objectid='2'.
:link reftype=hd refid=ddf1 auto split.
:h1 id=ddf1 x=50% y=top width=50% height=100% hide.ddf1
:ddf res=100.
```

The HM_QUERY_DDF_DATA message will be sent to the "GetName" window procedure, which can initialize and process the DDF data. Therefore, to incorporate DDF data in an online document, you must write a DLL to handle the processing. This DLL must be a global communication object. (That is, docproof-loaded)

The other DDF message is the HM_INVALIDATE_DDF_DATA. This message is sent by the application and informs IPF that previous dynamic data formatting (DDF) information is no longer valid.

For information about the DDF calls, see [Dynamic Data Formatting Functions](#).

Example using DDF

```
/* DDF Sample */

#define INCL_GPIPRIMITIVES
#define INCL_WINHELP
#define INCL_WIN
#define INCL_DDF
#include <os2.h>
#include "ddf.h"

MRESULT EXPENTRY ClientWndProc( HWND, USHORT, MPARAM, MPARAM );
VOID HelpInit( HAB hab );

HWND hwndHelpInstance;

HELPSUBTABLE helpSubTableMAIN[ ] =
{
    2, 0, 0
};

HELPTABLE helpTableMAIN[ ] =
{
    ID_PRIMWIN, helpSubTableMAIN, EXT_HELP_PANEL,
};

VOID main()
{
    static ULONG flFrameFlags = FCF_TITLEBAR | FCF_SYSMENU | FCF_TASKLIST |
                                FCF_SIZEBORDER | FCF_SHELLPOSITION |
                                FCF_MINBUTTON | FCF_MAXBUTTON;

    HAB hab;
    HMQ hmq;
    QMSG qmsg;
    HWND hwndFrame, hwndClient;
    BOOL bReturnCode;

    hab = WinInitialize( 0 );                /* Get anchor block */
```

```

hmq = WinCreateMsgQueue( hab, 0 );    /* Create message queue */

bReturnCode = WinRegisterClass( hab,          /* Anchor Block */
                                "ClientWindow", /* Class name */
                                (PFNWNDPROC) ClientWndProc, /* Window Proc */
                                CS_SIZEREDRAW,  /* Classstyles */
                                0 );           /* Extra data */

hwndFrame = WinCreateStdWindow( HWND_DESKTOP, /* parent */
                                WS_VISIBLE,   /* window styles */
                                &flFrameFlags, /* FCF values */
                                "ClientWindow", /* class */
                                "Press F1 for a DDF Sample", /* titlebar text */
                                0L,           /* client styles */
                                NULLHANDLE,   /* resource handle */
                                ID_PRIMWIN,   /* ID */
                                &hwndClient ); /* return client */

/* Initialize Help */
HelpInit( hab );
/* Associate the help instance with the window */
WinAssociateHelpInstance( hwndHelpInstance, hwndFrame );

while ( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) ) /* message loop */
    WinDispatchMsg( hab, &qmsg );

/* Destroy the help instance */
WinDestroyHelpInstance( hwndHelpInstance );

WinDestroyWindow( hwndFrame ); /* destroy window */
WinDestroyMsgQueue( hmq );     /* destroy message queue */
WinTerminate( hab );           /* return anchor block */
}

MRESULT EXPENTRY ClientWndProc( HWND hwnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{
    HPS hps;
    ULONG ulResID;
    Hddf hddf;
    HBITMAP hbm;
    APIRET returncode;
    ERRORID errcode;

    switch ( msg ) {
        case HM_INFORM:
            switch ( SHORT1FROMMP( mp1 ) ) {
                case 1:
                    WinMessageBox( HWND_DESKTOP, HWND_DESKTOP, "You clicked on the link",
                                   "DDF Sample", 0, MB_OK );
                    break;
                default:
                    WinMessageBox( HWND_DESKTOP, HWND_DESKTOP, "Unknown Inform link",
                                   "DDF Sample", 0, MB_OK );
                    break;
            } /* endswitch */
            break;
        case HM_QUERY_DDF_DATA:
            ulResID = LONGFROMMP( mp2 );
            /* Initialize DDF */
            hddf = DdfInitialize( hwndHelpInstance, 0L, 0L );
            /* Check res id to see which DDF line */
            switch ( ulResID ) {
                case 1:
                    DdfPara( hddf );
                    DdfText( hddf, "This text was placed by DDF" );
                    DdfPara( hddf );
                    DdfText( hddf, "You can use " );
                    DdfSetFontStyle( hddf, FM_SEL_ITALIC );
                    DdfText( hddf, "italic, " );
                    DdfSetFontStyle( hddf, FM_SEL_BOLD );
                    DdfText( hddf, "bold, " );
                    DdfSetFontStyle( hddf, FM_SEL_UNDERSCORE );
                    DdfText( hddf, "underscore, " );
                    DdfSetFontStyle( hddf, FM_SEL_UNDERSCORE | FM_SEL_ITALIC | FM_SEL_BOLD );
                    DdfText( hddf, "or all three!" );
                    DdfSetFontStyle( hddf, 0 );
                    DdfPara( hddf );
                    DdfText( hddf, "How about some color?" );
                    DdfPara( hddf );
                    DdfSetColor( hddf, CLR_BLUE, CLR_RED );
                    DdfText( hddf, "Red on Blue\n" );
                    DdfSetColor( hddf, CLR_BLACK, CLR_PALEGRAY );
                    DdfText( hddf, "Pale Gray on Black\n" );
                    DdfSetColor( hddf, CLR_DEFAULT, CLR_DEFAULT );
            }
    }
}

```

```

        DdfPara( hDDF);
        DdfSetFont( hDDF, "Courier", 100, 100);
        DdfText( hDDF, "Or a font change?");
        DdfSetFont( hDDF, NULL, 1, 1);
        DdfPara( hDDF);
        DdfSetTextAlign( hDDF, TA_CENTER);
        DdfSetFormat( hDDF, FALSE);
        DdfText( hDDF, "We can\ncenter text");
        DdfSetFormat( hDDF, TRUE);
        DdfSetTextAlign( hDDF, TA_LEFT);
        hbm = WinGetSysBitmap( HWND_DESKTOP, SBMP_FOLDER);
        DdfPara( hDDF);
        DdfBitmap( hDDF, hbm, ART_RUNIN );
        DdfText( hDDF, "Perhaps a bitmap? ");
        DdfPara( hDDF);
        DdfText( hDDF, "Or a list?");
        DdfBeginList( hDDF, 15, HMBT_FIT, HMLS7us.SINGLELELINE);
        DdfListItem( hDDF, "Item 1", "Item 1 Description");
        DdfListItem( hDDF, "Item 2", "Item 2 Description");
        DdfEndList( hDDF);
        DdfPara( hDDF);
        DdfText( hDDF, "You can even create a " );
        DdfHyperText( hDDF, "link to a panel ", "2", REFERENCE_BY_RES );
        DdfText( hDDF, "or an " );
        DdfInform( hDDF, "inform link", 1 );
        /* Return the DDF handle you just created */
        return (MRESULT) hDDF;
    break;
} /* endswitch */
break;
case WM_PAINT:
    hps = WinBeginPaint( hwnd, NULLHANDLE, NULL );
    GpiErase( hps );
    WinEndPaint( hps );
    return 0;
break;
} /* endswitch */
return WinDefWindowProc( hwnd, msg, mp1, mp2 );
}

VOID HelpInit( HAB hab )
{
    HELPINIT helpinit;

    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = NULL;
    helpinit.phtHelpTable = (PVOID) helpTableMAIN;
    helpinit.hmodHelpTableModule = 0;
    helpinit.hmodAccelActionBarModule = 0;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "Help for DDF Sample\0";
    helpinit.fShowPanelId = CMIC_HIDE_PANEL_ID;
    helpinit.pszHelpLibraryName = "DDF.HLP";

    hwndHelpInstance = WinCreateHelpInstance( hab, &helpinit );
    if (!hwndHelpInstance)
        WinMessageBox( HWND_DESKTOP, HWND_DESKTOP, "Help could not be initialized",
            "DDF Sample", 0, MB_OK | MB_ERROR);
    return;
}

```

Creating Master Indexes and Glossaries with Applications

The Workplace Shell "Mindex" software object class exploits IPF's functionality, providing a user object interface from which users can link to online helps. Your applications can instantiate these software object classes at run-time.

Master Help Index Objects and Glossary Objects

The OS/2 operating system's online documentation includes two Workplace Shell user objects that correspond to a single Workplace Shell software class object. The "Master Help Index" user object and the "Glossary" user object are instances of the "Mindex" object class. With the Master Help Index and the Glossary, users can access online help panels which include the **global** attribute in their index tagging (for more information, see [Customizing Master Help Index and Glossary Objects](#)). Note that this object class only works with online help (.HLP files), not online information (.INF files).

The Mindex class is a member of the WPAbstract class, which is, in turn, a member of the root WPObj class. All of the WPObj classes are easily accessed and instantiated through the "WP Class List", provided as one of the PM Development Tools in the *IBM Developer's Toolkit for OS/2*. For additional information about manipulating Mindex class user objects in the Workplace Shell (such as copying and shadowing) see [Customizing Master Help Index and Glossary Objects](#).

There are three task scenarios involving Mindex class objects of particular interest to a programmer:

1. Adding index entries to the Master Help Index user object at installation time.
2. Adding index entries to the Glossary user object at installation time.
3. Instantiating a new Mindex class object with an application. For example, creating a product-specific master index user object during the installation of your application.

These tasks can be done with minimal programming effort.

Adding Entries to the Master Help Index at Installation

Mindex class objects such as the Master Help Index and Glossary user objects display index entries that have been tagged with the **global** attribute (see [:i1.](#) and [:i2. \(Index\)](#)). These index entries can be placed in any .HLP file. The .HLP files available to a given Mindex class user object are identified in the object's Settings Notebook.

For example, the Master Help Index user object that is installed with the OS/2 operating system has a default "Files/Environment name(s)" setting of "HELP". This refers to the HELP environment variable identified in the CONFIG.SYS file. Therefore, if you wanted the **global** index entries in your .HLP files to appear in the Master Help Index user object, you could simply copy your .HLP files to directories or files specified by the HELP environment variable.

The default value of HELP is:

```
HELP=C:\OS2\HELP;C:\OS2\HELP\TUTORIAL;
```

Alternatively, you could modify the value of the "HELP" environment variable by updating CONFIG.SYS.

Adding Entries to the Glossary at Installation

Because both the Glossary and the Master Help Index user objects are simply Mindex class software objects, the manner in which you add entries to the Glossary is identical to the manner in which you add entries to the Master Help Index (see [Adding Entries to the Master Help Index at Installation](#)). However, if you check the default Settings Notebook for the Glossary user object, you will notice that the Glossary installed with the OS/2 operating system has a default "Files/Environment name(s)" that is not set to the "HELP" environment variable. Rather, it is set to the "GLOSSARY" environment variable. Therefore, you can update the value of this variable in CONFIG.SYS to reflect the location of your .HLP files that contain **global** tagged index entries. Alternatively, you can simply copy your .HLP files to the path defined by the GLOBAL environment variable. The default value is:

```
GLOSSARY=C:\OS2\HELP\GLOSS;
```

Creating New Mindex Class Objects

To instantiate a Mindex class object (such as a product-specific index or glossary), you need to create the new instance of the object and pass it necessary setup strings. This can be done with the *WinCreateObject* PM function.

If you need to create an instance of a Mindex class object for use by your application, you will probably want to do so during installation.

Example- Instantiating a Mindex Class Object Using REXX

Many applications are installed using the REXX procedural language. Rather than use the *WinCreateObject* function, you can use REXX's *SysCreateObject* function.

A sample REXX implementation might resemble the following:

```
/* Register and load the REXX Utility Functions */
call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs

/* Create a new Mindex class object */
/* Call SysCreateObject REXX Utility function, and pass it
   a class name (Mindex);
/* then pass it the icon title ('My Product Index');
/* then pass it the folder I want the object in (<WP_DESKTOP>;
/* then pass it the setup string with INDEX= set to the names
   of the help files that I want to be in my index
   (in this case C:\myprod\myhelp\myprod.hlp); the */
/* default value of INDEX is equal to the HELP
   environment variable;
/* then pass it a unique object id (MYINDEX);
/* then tell me if the installation was successful.
if SysCreateObject("Mindex", "My Product Index", "<WP_DESKTOP>",
   "INDEX=C:\MYPROD\HELP\MYPROD.HLP; OBJECTID=<MYINDEX>")
then Say 'Installation of My Product Index was successful!'
else Say 'Installation of My Product Index failed!'
```

Window Functions

Following is a summary of the window function calls that you would use to interface with IPF.

WinAssociateHelpInstance

Associates the help instance with the application window chain.

WinCreateHelpInstance

Calls the IPF help hook so that IPF can handle help requests.

WinCreateHelpTable

Identifies or changes the pointer to the help table in application memory.

WinDestroyHelpInstance

Ends the window chain's association with the help instance.

WinLoadHelpTable

Identifies or changes the handle of the module that contains the help table resource.

WinQueryHelpInstance

Identifies the help instance associated with a particular application window chain.

WinAssociateHelpInstance

WinAssociateHelpInstance - Syntax

This function associates the specified instance of the Help Manager with the window chain of the specified application window.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND     hwndHelpInstance; /* Handle of an instance of the Help Manager. */
HWND     hwndApp;          /* Handle of an application window. */
BOOL     rc;               /* Success indicator. */

rc = WinAssociateHelpInstance(hwndHelpInstance,
                             hwndApp);
```

WinAssociateHelpInstance Parameter - hwndHelpInstance

hwndHelpInstance (HWND) - input

Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

NULLHANDLE

Disassociates an instance of the Help Manager from a window chain when the instance has been destroyed.

Other

The handle of an instance of the Help Manager to be associated with the application window chain.

WinAssociateHelpInstance Parameter - hwndApp

hwndApp (HWND) - input

Handle of an application window.

The handle of the application window with which the instance of the Help Manager will be associated. The instance of the Help Manager is associated with the application window and any of its children or owned windows.

WinAssociateHelpInstance Return Value - rc

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinAssociateHelpInstance - Parameters

hwndHelpInstance (HWND) - input

Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

NULLHANDLE

Disassociates an instance of the Help Manager from a window chain when the instance has been destroyed.

Other

The handle of an instance of the Help Manager to be associated with the application window chain.

hwndApp (HWND) - input

Handle of an application window.

The handle of the application window with which the instance of the Help Manager will be associated. The instance of the Help Manager is associated with the application window and any of its children or owned windows.

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinAssociateHelpInstance - Remarks

In order to provide help, the application must associate an instance of the Help Manager with a chain of application windows. This association lets the Help Manager know which instance should provide the help function.

The Help Manager traces the window chain, starting from the window where help is requested. The application window in the chain with the associated help instance will be the one with which the Help Manager communicates and next to which the help window is positioned, unless a [HM_SET_ACTIVE_WINDOW](#) message is sent to the Help Manager. If the [HM_SET_ACTIVE_WINDOW](#) message is sent to the Help Manager, the active window parameter is the window with which the Help Manager communicates. The Help Manager positions the help window next to the window specified as the relative window.

WinAssociateHelpInstance - Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)
- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinAssociateHelpInstance - Related Messages

WinAssociateHelpInstance - Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION      100      /* id of HELP TABLE in resource file */

main( int argc, char *argv[], char *envp[] )
{
    HAB hab = WinInitialize( 0 );
    HMQ hmq = WinCreateMsgQueue( hab, 0 );
    HWND hwnd;
    HWND hwndClient;
    HWND hwndHelp;
    QMSG qmsg;
    ULONG flStyle;
    HELPINIT helpinit;

    /* Setup the help initialization structure */
    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = (PSZ)NULL;
    /* Help table in application resource */
    helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
    helpinit.hmodHelpTableModule = NULLHANDLE;
    /* Default action bar and accelerators */
    helpinit.hmodAccelActionBarModule = NULLHANDLE;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "APPNAME HELP";
    helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
    helpinit.pszHelpLibraryName = "APPNAME.HLP";

    /* Register the class */
    if( WinRegisterClass( ... ) )
    {
        /* create the main window */
        flStyle = FCF_STANDARD;
        hwnd = WinCreateStdWindow( ... );

        if( hwnd )
        {
            /* Create and associate the help instance */
            hwndHelp = WinCreateHelpInstance( hab, &helpinit );

            if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
            {
                /* Process messages */
                while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
                {
                    WinDispatchMsg( hab, &qmsg );
                } /* endwhile */
            }

            /* Remove help instance - note: add
            /* WinAssociateHelpInstance( NULLHANDLE, hwnd ); */
            /* to WM_DESTROY processing to remove the association. */
            WinDestroyHelpInstance( hwndHelp );
        }
    }

    /* finish the cleanup and exit */
    WinDestroyMsgQueue( hmq );
    WinTerminate( hab );
}
```

WinAssociateHelpInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinCreateHelpInstance

WinCreateHelpInstance - Syntax

This function creates an instance of the Help Manager with which to request Help Manager functions.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HAB          hab; /* Anchor-block handle. */  
PHELPINIT    phinitHMIInitStructure; /* Help Manager initialization structure. */  
HWND         hwndhelp; /* Help Manager handle. */  
  
hwndhelp = WinCreateHelpInstance(hab, phinitHMIInitStructure);
```

WinCreateHelpInstance Parameter - hab

hab (HAB) - input
Anchor-block handle.

The handle of the application anchor block returned from the WinInitialize function.

WinCreateHelpInstance Parameter - phinitHMIInitStructure

phinitHMIInitStructure (PHELPINIT) - in/out
Help Manager initialization structure.

WinCreateHelpInstance Return Value - hwndhelp

hwndhelp (HWND) - returns
Help Manager handle.

NULLHANDLE
Error occurred

Other
Help Manager handle.

WinCreateHelpInstance - Parameters

hab (HAB) - input
Anchor-block handle.

The handle of the application anchor block returned from the WinInitialize function.

phinitHMinitStructure (PHELPINIT) - in/out
Help Manager initialization structure.

hwndhelp (HWND) - returns
Help Manager handle.

NULLHANDLE
Error occurred

Other
Help Manager handle.

WinCreateHelpInstance - Remarks

If an error occurs, it is in the ulReturnCode parameter of the HELPINIT structure.

WinCreateHelpInstance - Related Functions

- [WinAssociateHelpInstance](#)
 - [WinCreateHelpInstance](#)
 - [WinCreateHelpTable](#)
 - [WinDestroyHelpInstance](#)
 - [WinLoadHelpTable](#)
 - [WinQueryHelpInstance](#)
-

WinCreateHelpInstance - Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL_WIN
#include <os2.h>
```

```

#define IDHT_APPLICATION      100      /* id of HELP TABLE in resource file
*/

main( int argc, char *argv[], char *envp[] )
{
    HAB hab = WinInitialize( 0 );
    HMQ hmq = WinCreateMsgQueue( hab, 0 );
    HWND hwnd;
    HWND hwndClient;
    HWND hwndHelp;
    QMSG qmsg;
    ULONG flStyle;
    HELPINIT helpinit;

    /* Setup the help initialization structure */
    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = (PSZ)NULL;
    /* Help table in application resource */
    helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
    helpinit.hmodHelpTableModule = NULLHANDLE;
    /* Default action bar and accelerators */
    helpinit.hmodAccelActionBarModule = NULLHANDLE;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "APPNAME HELP";
    helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
    helpinit.pszHelpLibraryName = "APPNAME.HLP";

    /* Register the class */
    if( WinRegisterClass( ... ) )
    {
        /* create the main window */
        flStyle = FCF_STANDARD;
        hwnd = WinCreateStdWindow( ... );

        if( hwnd )
        {
            /* Create and associate the help instance */
            hwndHelp = WinCreateHelpInstance( hab, &helpinit );

            if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
            {
                /* Process messages */
                while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
                {
                    WinDispatchMsg( hab, &qmsg );
                } /* endwhile */
            }

            /* Remove help instance - note: add
            /* WinAssociateHelpInstance( NULLHANDLE, hwnd ); */
            /* to WM_DESTROY processing to remove the association. */
            WinDestroyHelpInstance( hwndHelp );
        }
    }

    /* finish the cleanup and exit */
    WinDestroyMsgQueue( hmq );
    WinTerminate( hab );
}

```

WinCreateHelpInstance - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Example Code](#)

[Related Functions](#)

[Glossary](#)

WinCreateHelpTable

WinCreateHelpTable - Syntax

This function is used to identify or change the help table.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndHelpInstance; /* Handle of an instance of the Help Manager. */
PHELPTABLE phtHelpTable;    /* Help table allocated by the application. */
BOOL      rc;               /* Success indicator. */

rc = WinCreateHelpTable(hwndHelpInstance,
                        phtHelpTable);
```

WinCreateHelpTable Parameter - hwndHelpInstance

hwndHelpInstance (HWND) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

WinCreateHelpTable Parameter - phtHelpTable

phtHelpTable (PHELPTABLE) - input
Help table allocated by the application.

WinCreateHelpTable Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCreateHelpTable - Parameters

hwndHelpInstance (HWND) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

phtHelpTable (PHELPTABLE) - input
Help table allocated by the application.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

WinCreateHelpTable - Remarks

This function corresponds to the [HM_CREATE_HELP_TABLE](#) message that identifies a help table that is in memory.

WinCreateHelpTable - Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)
- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinCreateHelpTable - Related Messages

- [HM_CREATE_HELP_TABLE](#)

WinCreateHelpTable - Example Code

This example creates a help table in memory and passes the table to the Help Manager via WinCreateHelpTable. The help instance must have been created by WinCreateHelpInstance.

```
#define INCL_WINHELP
#include <os2.h>

/* DEFINES for window id's, menu items, controls, panels, etc. should */
/* be inserted here or in additional include files.                  */

/* Subtable for the main window's help */
```

```

HELPSUBTABLE phtMainTable[] = { 2,          /* Length of each entry */
                                /* Fill in one line for each menu item */
                                IDM_FILE,      PANELID_FILEMENU,
                                IDM_FILENEW,   PANELID_FILENEW,
                                IDM_FILEOPEN,  PANELID_FILEOPEN,
                                IDM_FILESAVE,  PANELID_FILESAVE,
                                IDM_FILESAVEAS, PANELID_FILESAVEAS,
                                IDM_FILEEXIT,  PANELID_FILEEXIT };

/* Subtable for the dialog window's help */
HELPSUBTABLE phtDlgTable[] = { 2,          /* Length of each entry */
                                /* Fill in one line for each control */
                                IDC_EDITFLD,   PANELID_DLGEDITFLD,
                                IDC_OK,        PANELID_DLGOK,
                                IDC_CANCEL,    PANELID_DLGCANCEL,
                                IDC_HELP,      PANELID_HELP };

/* Help table for the applications context sensitive help */
HELPTABLE phtHelpTable[] = { WINDOWID_MAIN, phtMainTable,
                              PANELID_MAINEXT,
                              WINDOWID_DLG,   phtDlgTable,   PANELID_DLGEEXT,
                              0,              NULL,           0 };

BOOL CreateHelpTable( HWND hWnd )
{
    BOOL bSuccess = FALSE;
    HWND hwndHelp;

    /* Get the associated help instance */
    hwndHelp = WinQueryHelpInstance( hWnd );

    if( hwndHelp )
    {
        /* Pass address of help table to the Help Manager */
        bSuccess = WinCreateHelpTable( hwndHelp, phtHelpTable );
    }

    /* return success indicator */
    return bSuccess;
}

```

WinCreateHelpTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinDestroyHelpInstance

WinDestroyHelpInstance - Syntax

This function destroys the specified instance of the Help Manager.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND    hwndHelpInstance; /* Handle of the instance of the Help Manager to be destroyed. */
BOOL    rc;               /* Success indicator. */

rc = WinDestroyHelpInstance(hwndHelpInstance);
```

WinDestroyHelpInstance Parameter - hwndHelpInstance

hwndHelpInstance (HWND) - input

Handle of the instance of the Help Manager to be destroyed.

This is the handle returned by the [WinCreateHelpInstance](#) call.

WinDestroyHelpInstance Return Value - rc

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinDestroyHelpInstance - Parameters

hwndHelpInstance (HWND) - input

Handle of the instance of the Help Manager to be destroyed.

This is the handle returned by the [WinCreateHelpInstance](#) call.

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinDestroyHelpInstance - Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)

- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinDestroyHelpInstance - Example Code

This example shows a typical main function for an application which uses help. Following creation of the main application window the Help Manager is initialized and associated with the window. The help table is defined in the application's resources. When the window is destroyed, terminating the application, the help instance is also destroyed.

```
#define INCL=_WIN
#include <os2.h>

#define IDHT_APPLICATION      100      /* id of HELP TABLE in resource file
*/

main( int argc, char *argv[], char *envp[] )
{
    HAB  hab = WinInitialize( 0 );
    HMQ  hmq = WinCreateMsgQueue( hab, 0 );
    HWND hwnd;
    HWND hwndClient;
    HWND hwndHelp;
    QMSG qmsg;
    ULONG flStyle;
    HELPINIT helpinit;

    /* Setup the help initialization structure */
    helpinit.cb = sizeof( HELPINIT );
    helpinit.ulReturnCode = 0L;
    helpinit.pszTutorialName = (PSZ)NULL;
    /* Help table in application resource */
    helpinit.phtHelpTable = (PHELPTABLE)MAKEULONG( IDHT_APPLICATION, 0xffff );
    helpinit.hmodHelpTableModule = NULLHANDLE;
    /* Default action bar and accelerators */
    helpinit.hmodAccelActionBarModule = NULLHANDLE;
    helpinit.idAccelTable = 0;
    helpinit.idActionBar = 0;
    helpinit.pszHelpWindowTitle = "APPNAME HELP";
    helpinit.fShowPanelId = CMIC_SHOW_PANEL_ID;
    helpinit.pszHelpLibraryName = "APPNAME.HLP";

    /* Register the class */
    if( WinRegisterClass( ... ) )
    {
        /* create the main window */
        flStyle = FCF_STANDARD;
        hwnd = WinCreateStdWindow( ... );

        if( hwnd )
        {
            /* Create and associate the help instance */
            hwndHelp = WinCreateHelpInstance( hab, &helpinit );

            if( hwndHelp && WinAssociateHelpInstance( hwndHelp, hwnd ) )
            {
                /* Process messages */
                while( WinGetMsg( hab, &qmsg, NULLHANDLE, 0, 0 ) )
                {
                    WinDispatchMsg( hab, &qmsg );
                } /* endwhile */
            }

            /* Remove help instance - note: add */
            /* WinAssociateHelpInstance( NULLHANDLE, hwnd ); */
            /* to WM_DESTROY processing to remove the association. */
            WinDestroyHelpInstance( hwndHelp );
        }
    }

    /* finish the cleanup and exit */
    WinDestroyMsgQueue( hmq );
    WinTerminate( hab );
}
```

WinDestroyHelpInstance - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

WinLoadHelpTable

WinLoadHelpTable - Syntax

This function identifies the module handle and identity of the help table to the instance of the Help Manager.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */
#include <os2.h>

HWND      hwndHelpInstance; /* Handle of an instance of the Help Manager. */
ULONG     idHelpTable;      /* Identity of the help table. */
HMODULE    Module;          /* Handle of the module which contains the help table and help subtable resources */
BOOL      rc;               /* Success indicator. */

rc = WinLoadHelpTable(hwndHelpInstance, idHelpTable,
    Module);
```

WinLoadHelpTable Parameter - hwndHelpInstance

hwndHelpInstance (HWND) - input
Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

WinLoadHelpTable Parameter - idHelpTable

idHelpTable (ULONG) - input
Identity of the help table.

It must be greater or equal to 0 and less or equal to 0xFFFF.

WinLoadHelpTable Parameter - Module

Module (HMODULE) - input

Handle of the module which contains the help table and help subtable resources.

NULLHANDLE

Use the resources file for the application.

Other

The module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the help resources.

WinLoadHelpTable Return Value - rc

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinLoadHelpTable - Parameters

hwndHelpInstance (HWND) - input

Handle of an instance of the Help Manager.

This is the handle returned by the [WinCreateHelpInstance](#) call.

idHelpTable (ULONG) - input

Identity of the help table.

It must be greater or equal to 0 and less or equal to 0xFFFF.

Module (HMODULE) - input

Handle of the module which contains the help table and help subtable resources.

NULLHANDLE

Use the resources file for the application.

Other

The module handle returned by the DosLoadModule or DosQueryModuleHandle call referencing a dynamic-link library containing the help resources.

rc (BOOL) - returns

Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

WinLoadHelpTable - Remarks

An application specifies or changes the handle of the module which contains the help table or the identity of the help table.

This function corresponds to the [HM_LOAD_HELP_TABLE](#) message that identifies the identifier of a help table and the handle of the module which contains the help table and its associated help subtables.

WinLoadHelpTable - Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)
- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinLoadHelpTable - Related Messages

- [HM_LOAD_HELP_TABLE](#)

WinLoadHelpTable - Example Code

```
BOOL LoadHelpTable( HWND hWnd, USHORT usResource, PSZ pszModuleName )
{
    BOOL bSuccess = FALSE;
    HMODULE hmodule;
    HWND hwndHelp;
    PSZ pszObjNameBuf[ 80 ];

    /* Get the DLL loaded */
    if( !DosLoadModule( pszObjNameBuf, sizeof( pszObjNameBuf ),
                      pszModuleName, &hmodule ) )
    {
        /* Get the associated help instance */
        hwndHelp = WinQueryHelpInstance( hWnd );

        if( hwndHelp )
        {
            /* Pass address of help table to the Help Manager */
            bSuccess = WinLoadHelpTable( hwndHelp, usResource, hmodule );
        }
    }

    /* Return success indicator */
    return bSuccess;
}
```

WinLoadHelpTable - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Related Messages](#)
[Glossary](#)

WinQueryHelpInstance

WinQueryHelpInstance - Syntax

This function enables the application to query the instance of the Help Manager associated with the application-supplied window handle.

```
#define INCL_WINHELP /* Or use INCL_WIN, INCL_PM, */  
#include <os2.h>  
  
HWND    hwndApp; /* Handle of the application window. */  
HWND    hwndHelp; /* Help Manager window handle. */  
  
hwndHelp = WinQueryHelpInstance(hwndApp);
```

WinQueryHelpInstance Parameter - hwndApp

hwndApp (HWND) - input
Handle of the application window.

WinQueryHelpInstance Return Value - hwndHelp

hwndHelp (HWND) - returns
Help Manager window handle.

NULLHANDLE	No Help Manager instance is associated with the application window.
Other	Help Manager window handle.

WinQueryHelpInstance - Parameters

hwndApp (HWND) - input
Handle of the application window.

hwndHelp (HWND) - returns
Help Manager window handle.

NULLHANDLE

No Help Manager instance is associated with the application window.

Other

Help Manager window handle.

WinQueryHelpInstance - Remarks

The Help Manager first traces the parent window chain until it is NULLHANDLE or HWND_DESKTOP. Then Help Manager traces the owner chain. If a parent of the owner window exists, the trace begins again with the parent chain.

The window chain will be traced until the Help Manager finds an instance of the Help Manager or until both the parent and owner windows are NULLHANDLE or HWND_DESKTOP.

WinQueryHelpInstance - Related Functions

- [WinAssociateHelpInstance](#)
- [WinCreateHelpInstance](#)
- [WinCreateHelpTable](#)
- [WinDestroyHelpInstance](#)
- [WinLoadHelpTable](#)
- [WinQueryHelpInstance](#)

WinQueryHelpInstance - Example Code

This example shows the use of the WinQueryHelpInstance call during the processing of a WM_INITMENU message in order to obtain the handle for sending an HM_SET_ACTIVE_WINDOW message.

```
#define INCL_WIN
#include <os2.h>

MRESULT wm_initmenu( HWND hWnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    /* Send message to establish the current window's parent          */
    /* as the active help window.                                     */
    WinSendMsg( WinQueryHelpInstance( hWnd ),
                HM_SET_ACTIVE_WINDOW,
                (MPARAM)WinQueryWindow( hWnd, QW_PARENT ),
                (MPARAM)WinQueryWindow( hWnd, QW_PARENT ) );

    /* Pass message on for default processing                         */
    return WinDefWindowProc( hWnd, ulMsg, mp1, mp2 );
}
```

WinQueryHelpInstance - Topics

Select an item:

Dynamic Data Formatting Functions

The application can also use the window to establish a dialog with the user, and format text responses in the window by calling dynamic data formatting (DDF) routines. The DDF functions provide limited formatting of text at run time.

Warning: Users can not print or search DDF data.

Following is a summary of the DDF calls that you can use in your Presentation Manager application.

[DdfBeginList](#)

Begins a definition list in the DDF buffer.

[DdfBitmap](#)

Places a reference to a bit map in the DDF buffer.

[DdfEndList](#)

Terminates the definition list initialized by DdfBeginList.

[DdfHyperText](#)

Defines a hypertext link to another window.

[DdfInform](#)

Defines a hypertext inform link.

[DdfInitialize](#)

Initializes the IPF internal structures for a DDF facility and returns a DDF handle.

[DdfListItem](#)

Inserts a definition list entry item in the DDF buffer.

[DdfMetafile](#)

Places a reference to a metafile into the DDF buffer.

[DdfPara](#)

Creates a paragraph within the DDF buffer.

[DdfSetColor](#)

Sets the background and foreground colors of the displayed text.

[DdfSetFont](#)

Specifies a text font (Courier) in the DDF buffer.

[DdfSetFontStyle](#)

Specifies a text font (bold face) in the DDF buffer.

[DdfSetFormat](#)

Turns formatting off or on.

[DdfSetTextAlign](#)

Defines whether left, center, or right text justification is to be used when text formatting is off.

[DdfText](#)

Adds text to the DDF buffer.

DdfBeginList

DdfBeginList - Syntax

This function begins a definition list in the DDF buffer; it corresponds to the **:dl.** (definition list) tag.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
ULONG     ulWidthDT;     /* Width of the definition term. */
ULONG     fBreakType;    /* Type of line break to use. */
ULONG     fSpacing;      /* Spacing between definitions. */
BOOL      rc;            /* Success indicator. */

rc = DdfBeginList(hddf, ulWidthDT, fBreakType,
                  fSpacing);
```

DdfBeginList Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfBeginList Parameter - ulWidthDT

ulWidthDT (ULONG) - input
Width of the definition term.

DdfBeginList Parameter - fBreakType

fBreakType (ULONG) - input
Type of line break to use.

The following constants may be specified:

- | | |
|-----------|---|
| HMBT_ALL | Start all definition descriptions on the next line, regardless of the actual lengths of definition terms. |
| HMBT_FIT | Start definition description on the next line only when the definition term is longer than the width specified. |
| HMBT_NONE | Do not start the definition description on the next line, even when the definition term is longer than the width specified. |
-

DdfBeginList Parameter - fSpacing

fSpacing (ULONG) - input
Spacing between definitions.

Only the following constants may be specified:

HMLS_SINGLELINE
Do not insert a blank line between each definition description and the next definition term.

HMLS_DOUBLELINE
Insert a blank line between each definition description and the next definition term.

DdfBeginList Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE
Successful completion

FALSE
Error occurred.

DdfBeginList - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

ulWidthDT (ULONG) - input
Width of the definition term.

fBreakType (ULONG) - input
Type of line break to use.

The following constants may be specified:

HMBT_ALL
Start all definition descriptions on the next line, regardless of the actual lengths of definition terms.

HMBT_FIT
Start definition description on the next line only when the definition term is longer than the width specified.

HMBT_NONE
Do not start the definition description on the next line, even when the definition term is longer than the width specified.

fSpacing (ULONG) - input
Spacing between definitions.

Only the following constants may be specified:

HMLS_SINGLELINE
Do not insert a blank line between each definition description and the next definition term.

HMLS_DOUBLELINE
Insert a blank line between each definition description and the next definition term.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfBeginList - Remarks

Once this function has been called, use of any DDF function other than [DdfListItem](#), [DdfSetColor](#), and [DdfEndList](#) may produce unpredictable results.

DdfBeginList - Errors

Possible returns from [WinGetLastError](#)

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_LIST_UNCLOSED (0x3007)
An attempt was made to nest a list.

HMERR_DDF_LIST_BREAKTYPE (0x3009)
The value of BreakType is not valid.

HMERR_DDF_LIST_SPACING (0x300A)
The value for Spacing is not valid.

DdfBeginList - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfBeginList - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfBeginList](#) to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```

#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM              /* definition list */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{ "MVS", "Multiple Virtual System"},
                  { "VM", "Virtual Machine"} };

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{
    HWND    hwndParent;
    HWND    hwndInstance;
    Hddf    hDdf;          /* DDF handle */
    SHORT   i;             /* loop index */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,          /* Default buffer size */
            0L,          /* Default increment */
        );

        if (hDdf == NULLHANDLE) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                             Definition[i].Desc))
            {
                return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}

```

DdfBeginList - Topics

Select an item:

[Syntax](#)

[Parameters](#)

DdfBitmap

DdfBitmap - Syntax

This function places a reference to a bit map in the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;    /* Handle to DDF returned by DdfInitialize. */
HBITMAP    hbm;    /* Standard Presentation Manager bit map handle. */
ULONG      fAlign; /* Text justification flag. */
BOOL       rc;     /* Success indicator. */

rc = DdfBitmap(hddf, hbm, fAlign);
```

DdfBitmap Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfBitmap Parameter - hbm

hbm (HBITMAP) - input
Standard Presentation Manager bit map handle.

DdfBitmap Parameter - fAlign

fAlign (ULONG) - input
Text justification flag.

Any of the following values can be specified:

ART_LEFT	Left-justify the bit map.
ART_RIGHT	Right-justify the bit map.
ART_CENTER	Center the bit map.
ART_RUNIN	Allow the bit map to be reflowed with text.

DdfBitmap Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfBitmap - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

hbm (HBITMAP) - input
Standard Presentation Manager bit map handle.

fAlign (ULONG) - input
Text justification flag.

Any of the following values can be specified:

ART_LEFT	Left-justify the bit map.
ART_RIGHT	Right-justify the bit map.
ART_CENTER	Center the bit map.
ART_RUNIN	Allow the bit map to be reflowed with text.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfBitmap - Remarks

The handle to the presentation space in which the bit map was created cannot be freed by the application while the panel is displayed.

Note: There is a (3-byte + size of HBITMAP structure). ESC code overhead in the DDF internal buffer for this function. There is a 1-byte ESC code overhead required for the *fAlign* flag.

DdfBitmap - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_ALIGN_TYPE (0x3002)
The alignment type is not valid.

DdfBitmap - Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfBitmap - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example obtains a device context (DevOpenDC), creates a presentation space (GpiCreatePS), and loads a bit map (GpiLoadBitmap). It then uses DdfBitmap to place a reference to the bit map in the DDF buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_GPICONTROL        /* Basic PS control */
#define INCL_GPIBITMAPS        /* Bit maps and Pel Operations */
#define INCL_GPIPRIMITIVES     /* Drawing Primitives/Attributes */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

#define ACVP_HAB 12
#define BM_HPS 16
#define BM_HDC 20
#define BM_HWND 24
#define ID_LEFT 255

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;    /* parent window */
    HWND    hwndInstance;  /* help instance window */
    Hddf     hDdf;          /* DDF handle */
    HDC      hdc;           /* device context handle */
    HPS      hps;           /* presentation space handle */
    ...
}
```

```

HAB      hab;                /* anchor block handle                */
SIZEL    sizel = {0L,0L};    /* size of new PS                */
HBITMAP  hBitmap;           /* bit map handle                */
HMODULE  hModule;           /* module handle                */

switch( ulMsg )
{
case HM_QUERY_DDF_DATA:
    hwndParent = WinQueryWindow( hwnd, QW_PARENT );
    hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
    hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                     MPFROMSHORT( HMQW_INSTANCE ), NULL );

    /* Allocate 1K Buffer (default) */
    hDdf = DdfInitialize(
        hwndInstance, /* Handle of help instance */
        0L,           /* Default buffer size */
        0L,           /* Default increment */
    );

    if (hDdf == NULLHANDLE) /* Check return code */
    {
        return (MRESULT)FALSE;
    }

    /* get module handle for bit map */
    DosQueryModuleHandle("bitmap", &hModule);
    if (hModule == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* get hab for this window */
    if ((hab = (HAB)WinQueryWindowULong(hwnd, ACVP_HAB)) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* create a device context */
    if ((hdc = DevOpenDC(hab, OD_MEMORY, "", 0L,
                        (PDEVOPENDATA)NULL, (HDC)NULL)) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* save hdc in reserved word */
    WinSetWindowULong(hwnd, BM_HDC, (ULONG)hdc);

    /* create a noncached micro presentation space */
    /* and associate it with the window */
    if ((hps = GpiCreatePS(hab, hdc, &sizel,
                          PU_PELS | GDI_DEFAULT
                          | GPIT_MICRO | GPIA_ASSOC)) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* save hps in reserved word */
    WinSetWindowULong(hwnd, BM_HPS, (ULONG)hps);

    /* Load the Bit map to display */
    if ((hBitmap = GpiLoadBitmap(hps, hModule, ID_LEFT, 300L,
                                300L)) == NULLHANDLE)
    {
        return (MRESULT)FALSE;
    }

    /* save bit map hwnd in reserved word */
    WinSetWindowULong(hwnd, BM_HWND, (ULONG)hBitmap);

    /* Display the bit map align left */
    if (!DdfBitmap(hDdf, hBitmap, ART_LEFT))
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;

case WM_CLOSE:
    /* release PS, DC, and bit map */
    GpiDestroyPS((HPS)WinQueryWindowULong(hwnd, BM_HPS));
    DevCloseDC((HDC)WinQueryWindowULong(hwnd, BM_HDC));
    GpiDeleteBitmap((HBITMAP)WinQueryWindowULong(hwnd, BM_HWND));

```

```
        WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT));  
        return (MRESULT)TRUE;  
    }  
}
```

DdfBitmap - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfEndList

DdfEndList - Syntax

This function terminates the definition list initialized by DdfBeginList.

```
#define INCL_DDF  
#include <os2.h>  
  
HDDF    hddf; /* Handle to DDF returned by DdfInitialize. */  
BOOL    rc;   /* Success indicator. */  
  
rc = DdfEndList(hddf);
```

DdfEndList Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfEndList Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfEndList - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfEndList - Errors

Possible returns from WinGetLastError

HMERR_DDF_LIST_UNINITIALIZED (0x3008)
No definition list has been initialized by [DdfBeginList](#).

DdfEndList - Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfEndList - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfBeginList](#) to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```

#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM              /* definition list */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{ "MVS", "Multiple Virtual System"},
                  { "VM", "Virtual Machine"} };

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{
    HWND    hwndParent;
    HWND    hwndInstance;
    Hddf    hDdf;          /* DDF handle */
    SHORT   i;             /* loop index */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,           /* Default buffer size */
            0L,           /* Default increment */
        );

        if (hDdf == NULLHANDLE) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                             Definition[i].Desc))
            {
                return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}

```

DdfEndList - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)

DdfHyperText

DdfHyperText - Syntax

This function defines a hypertext link to another panel, which is equal to a link of reftype=hd. Links to footnotes, launch links and links to external databases are not supported.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;           /* Handle to DDF returned by DdfInitialize. */
PCSZ      pszText;        /* Hypertext phrase. */
PCSZ      pszReference;    /* Pointer to a string containing the link reference. */
ULONG     fReferenceType; /* Reference Type. */
BOOL      rc;             /* Success indicator. */

rc = DdfHyperText(hddf, pszText, pszReference,
                  fReferenceType);
```

DdfHyperText Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfHyperText Parameter - pszText

pszText (PCSZ) - input
Hypertext phrase.

DdfHyperText Parameter - pszReference

pszReference (PCSZ) - input
Pointer to a string containing the link reference.

The value of this parameter depends on the value of *fReferenceType*.

If *fReferenceType* is REFERENCE_BY_RES

pszReference must contain a pointer to a numeric string containing the res number; otherwise it will default to a res number of zero. Valid values are 1 - 64000; all other values are reserved.

If *fReferenceType* is REFERENCE_BY_ID

pszReference must contain a pointer to a string containing the alphanumeric identifier of the destination panel.

DdfHyperText Parameter - fReferenceType

fReferenceType (ULONG) - input
Reference Type.

This parameter specifies whether you are linking via a resource identifier (res number) or via an alphanumeric identifier.

REFERENCE_BY_RES

To link via a resource identifier.

REFERENCE_BY_ID

To link via an alphanumeric identifier.

DdfHyperText Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfHyperText - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

pszText (PCSZ) - input
Hypertext phrase.

pszReference (PCSZ) - input
Pointer to a string containing the link reference.

The value of this parameter depends on the value of *fReferenceType*.

If *fReferenceType* is REFERENCE_BY_RES

pszReference must contain a pointer to a numeric string containing the res number; otherwise it will default to a res number of zero. Valid values are 1 - 64000; all other values are reserved.

If *fReferenceType* is REFERENCE_BY_ID

pszReference must contain a pointer to a string containing the alphanumeric identifier of the destination panel.

fReferenceType (ULONG) - input
Reference Type.

This parameter specifies whether you are linking via a resource identifier (res number) or via an alphanumeric identifier.

REFERENCE_BY_RES
To link via a resource identifier.
REFERENCE_BY_ID
To link via an alphanumeric identifier.

rc (BOOL) - returns
Success indicator.

TRUE
Successful completion
FALSE
Error occurred.

DdfHyperText - Remarks

There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character. If *fReferenceType* is REFERENCE_BY_ID, then there is a (3-byte + Reference length) ESC code overhead. For a *fReferenceType* of REFERENCE_BY_RES, the overhead is 5 bytes. Finally, there is a 3-byte ESC code overhead that is required for ending the hypertext link.

DdfHyperText - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_REFTYPE (0x3006)
The reference type is not valid.

DdfHyperText - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfHyperText - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses DdfHyperText to create a hypertext link with another resource. For a more

detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

PSZ   Text = "This text is a HYPERTEXT message.\n"; /* hypertext string */
PSZ   ResID = "1"; /* Resource identifier */

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND   hwndParent;
    HWND   hwndInstance;
    Hddf    hDdf; /* DDF handle */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,           /* Default buffer size */
            0L,           /* Default increment */
        );

        if (hDdf == NULLHANDLE) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* create hypertext link with resource 1 */
        if (!DdfHyperText(hDdf, (PSZ)Text, ResID, REFERENCE_BY_RES))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfHyperText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfInform

DdfInform - Syntax

This function defines a hypertext inform link, which sends an HM_INFORM message to the application with a **res=** attribute. It also corresponds to the link tag with retype=inform.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
PCSZ      pszText;       /* Hypertext phrase. */
ULONG     resInformNumber; /* Res number associated with this hypertext field. */
BOOL      rc;            /* Success indicator. */

rc = DdfInform(hddf, pszText, resInformNumber);
```

DdfInform Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfInform Parameter - pszText

pszText (PCSZ) - input
Hypertext phrase.

DdfInform Parameter - resInformNumber

resInformNumber (ULONG) - input
Res number associated with this hypertext field.

Possible values are 1 to 64000; all other values are reserved.

DdfInform Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfInform - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

pszText (PCSZ) - input
Hypertext phrase.

resInformNumber (ULONG) - input
Res number associated with this hypertext field.

Possible values are 1 to 64000; all other values are reserved.

rc (BOOL) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfInform - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfInform - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfInform - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses DdfInform to create a hypertext inform link with another resource (corresponds to the IPF :link. tag with reftype=inform). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#)

sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

PSZ Text = "This text is a HYPERTEXT message.\n"; /* hypertext string */
MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;
    Hddf     hDdf;          /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,          /* Default buffer size */
                0L,          /* Default increment */
            );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* create hypertext inform link with resource 1 */
            if (!DdfInform(hDdf, (PSZ)Text, 1L))
            {
                return (MRESULT)FALSE;
            }

            return (MRESULT)hDdf;
        }
    }
}
```

DdfInform - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfInitialize

DdfInitialize - Syntax

This function initializes the IPF internal structures for dynamic data formatting and returns a DDF handle. The application uses this handle to refer to a particular DDF panel.

```
#define INCL_DDF
#include <os2.h>

HWND      hwndHelpInstance; /* Handle of a help instance. */
ULONG     cbBuffer;         /* Initial length of internal buffer. */
ULONG     ulIncrement;      /* Amount by which to increment the buffer size, if necessary. */
HDDF      hddf;             /* Handle to DDF. */

hddf = DdfInitialize(hwndHelpInstance, cbBuffer,
                     ulIncrement);
```

DdfInitialize Parameter - hwndHelpInstance

hwndHelpInstance (HWND) - input
Handle of a help instance.

DdfInitialize Parameter - cbBuffer

cbBuffer (ULONG) - input
Initial length of internal buffer.

Initial length of internal buffer where DDF information is to be stored. If this field is 0L, a default value of 1K is defined. The maximum value is 61 440 bytes (60KB).

DdfInitialize Parameter - ulIncrement

ulIncrement (ULONG) - input
Amount by which to increment the buffer size, if necessary.

If this field is NULL, a default value of 256 bytes is defined. The maximum value is 60KB.

DdfInitialize Return Value - hddf

hddf (HDDF) - returns
Handle to DDF.

A handle to DDF is returned if initialization was successful. Otherwise, the value returned is NULL, indicating that an error has occurred because of insufficient memory or incorrect instance.

DdfInitialize - Parameters

hwndHelpInstance (HWND) - input
Handle of a help instance.

cbBuffer (ULONG) - input
Initial length of internal buffer.

Initial length of internal buffer where DDF information is to be stored. If this field is 0L, a default value of 1K is defined. The maximum value is 61 440 bytes (60KB).

ullIncrement (ULONG) - input
Amount by which to increment the buffer size, if necessary.

If this field is NULL, a default value of 256 bytes is defined. The maximum value is 60KB.

hddf (HDDF) - returns
Handle to DDF.

A handle to DDF is returned if initialization was successful. Otherwise, the value returned is NULL, indicating that an error has occurred because of insufficient memory or incorrect instance.

DdfInitialize - Remarks

At initialization, the default for dynamic data display is that text aligned on the left, and formatting is turned on.

DdfInitialize - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfInitialize - Example Code

This example shows how to initialize and use the Dynamic Data Facility for displaying an online document. Two functions are defined: the first, `SampleObj`, creates a window that will display the online information and specifies the second function, `SampleWindowProc`, as the corresponding window procedure. These two functions are compiled into a DLL and exported, so that IPF can invoke them when it encounters the `:ddf` and `:acviewport` tags during execution. The `:acviewport` tag will specify the DLL name and the `SampleObj` function; when IPF calls `SampleObj`, it initializes an application-controlled window with `SampleWindowProc` as the window procedure and returns the window handle. Later, when IPF encounters the `:ddf` tag, it will send `SampleWindowProc` an `HM_QUERY_DDF_DATA` message. At this point, before calling

any of the DDF API, DdfInitialize must first be called to initiate a DDF buffer, after which the other DDF API can be called to display the online information.

```
#define INCL_WINWINDOWMGR          /* General window management */
#define INCL_WINMESSAGEMGR        /* Message management */
#define INCL_WINDIALOGS           /* Dialog boxes */
#define INCL_DDF                  /* Dynamic Data Facility */
#define INCL_32
#include <os2.h>
#include <pmhelp.h>

#define COM_HWND 4                  /* window word offsets */
#define PAGE_HWND 8
#define ACVP_HAB 12

USHORT DdfClass = FALSE;

MRESULT EXPENTRY SampleWindowProc(HWND hWnd, ULONG Message,
                                   MPARAM lParam1, MPARAM lParam2);

USHORT APIENTRY SampleObj(PACVP pACVP, PCH Parameter)
{
    HWND DdfHwnd;                  /* Client window handle */
    HWND DdfCHwnd;                 /* Child window handle */
    HWND PreviousHwnd;             /* Handle for setting comm window active */

    /* register DDF Base class if not registered already */
    if (!DdfClass)
    {
        if (!WinRegisterClass(
            pACVP->hAB,              /* Anchor block handle */
            "CLASS_Ddf",            /* Application window class name */
            SampleWindowProc,       /* Address of window procedure */
            CS_SYNCPAINT |          /* Window class style */
            CS_SIZEREDRAW |
            CS_MOVEENOTIFY,         /* Extra storage */
            20))
        {
            return TRUE;
        }
        DdfClass = TRUE;
    }

    /* create standard window */
    if (!(DdfHwnd = WinCreateStdWindow(
        pACVP->hWndParent,          /* ACVP is parent */
        0L,                        /* No class style */
        NULL,                      /* Frame control flag */
        "CLASS_Ddf",              /* Window class name */
        NULL,                      /* No title bar */
        0L,                       /* No special style */
        0L,                       /* Resource in .EXE */
        0,                         /* No window identifier */
        &DdfCHwnd )))             /* Client window handle */
    {
        return FALSE;
    }

    /* store the frame window handle in ACVP data structure */
    pACVP->hWndACVP = DdfHwnd;

    /* set this window as active communication window */
    PreviousHwnd = (HWND)WinSendMsg(pACVP->hWndParent,
                                    HM_SET_OBJCOM_WINDOW,
                                    MPFROMHWORD(DdfHwnd), NULL);

    /* save returned communication hwnd in reserved word */
    WinSetWindowULong(DdfCHwnd, COM_HWND, (ULONG)PreviousHwnd);

    /* save anchor block handle in reserved word */
    WinSetWindowULong(DdfCHwnd, ACVP_HAB, (ULONG)pACVP->hAB);

    return FALSE;
} /* SampleObj */

MRESULT EXPENTRY SampleWindowProc(HWND hWnd, ULONG Message,
                                   MPARAM lParam1, MPARAM lParam2)
{
    HWND hwnParent;                /* parent window */
    HWND hwnHelpInstance;          /* help instance window */
    Hddf hDdf;                     /* DDF handle */
    ULONG DdfID;                   /* DDF resource id */

```



```

switch (Message)
{
case HM_QUERY_DDF_DATA:
    WinSetWindowULong(hWnd, PAGE_HWND, LONGFROMMP(lParam1));
    DdfID = LONGFROMMP(lParam2);
    hwndParent = WinQueryWindow(hWnd, QW_PARENT);
    hwndParent = WinQueryWindow(hwndParent, QW_PARENT);
    hwndHelpInstance = (HWND)WinSendMsg(hwndParent, HM_QUERY,
        MPFROMSHORT(HMQW_INSTANCE), NULL);

    /* Allocate 1K Buffer (default) */
    hDdf = DdfInitialize(
        hwndHelpInstance, /* Handle of help instance */
        0L,                /* Default buffer size */
        0L,                /* Default increment */
    );

    if (hDdf == NULLHANDLE) /* Check return code */
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;

default:
    return (WinDefWindowProc(hWnd, Message, lParam1, lParam2));
}
} /* SampleWindowProc */

```

DdfInitialize - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfListItem

DdfListItem - Syntax

This function inserts a definition list entry in the DDF buffer; it corresponds to a combination of the **:dt.** (definition term) and **:dd.** (definition define) tags.

```

#define INCL_DDF
#include <os2.h>

HDDF    hddf;          /* Handle to DDF returned by DdfInitialize. */
PCSZ    pszTerm;        /* Term portion of the definition list entry. */
PCSZ    pszDescription; /* Description portion of the definition list entry. */
BOOL    rc;            /* Success indicator. */

rc = DdfListItem(hddf, pszTerm, pszDescription);

```

DdfListItem Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfListItem Parameter - pszTerm

pszTerm (PCSZ) - input
Term portion of the definition list entry.

DdfListItem Parameter - pszDescription

pszDescription (PCSZ) - input
Description portion of the definition list entry.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in both the term and the description. There is a 1-byte ESC code overhead for each blank and for each newline character. For each list item, there is a 5-byte ESC code overhead for the margin alignment.

DdfListItem Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfListItem - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

pszTerm (PCSZ) - input
Term portion of the definition list entry.

pszDescription (PCSZ) - input
Description portion of the definition list entry.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in both the term and the description. There is a 1-byte ESC code overhead for each blank and for each newline character. For each list item, there is a 5-byte ESC code overhead for the margin alignment.

rc (BOOL) - returns
Success indicator.

TRUE
Successful completion
FALSE
Error occurred.

DdfListItem - Remarks

The handle to the presentation space in which the bit map was created cannot be freed by the application while the panel is displayed.

Note: There is a (3-byte + size of HBITMAP structure) ESC code overhead in the DDF internal buffer for this function. There is a 1-byte ESC code overhead required for the Align flag.

DdfListItem - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_LIST_UNINITIALIZED (0x3008)
No definition list has been initialized by [DdfBeginList](#).

DdfListItem - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfListItem - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfBeginList](#) to indicate the beginning of a definition list in the DDF buffer (this corresponds to the IPF dl tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

struct _LISTITEM              /* definition list */
{
    PSZ Term;
    PSZ Desc;
} Definition[2] = {{ "MVS", "Multiple Virtual System"},
                  { "VM", "Virtual Machine"} };

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2)
{
    HWND    hwndParent;
    HWND    hwndInstance;
    HDDF    hDdf;          /* DDF handle */
    SHORT i;               /* loop index */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,          /* Default buffer size */
            0L,          /* Default increment */
        );

        if (hDdf == NULLHANDLE) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* begin definition list */
        if (!DdfBeginList(hDdf, 3L, HMBT_ALL, HMLS_SINGLELINE))
        {
            return (MRESULT)FALSE;
        }

        /* insert 2 entries into definition list */
        for (i=0; i < 2; i++)
        {
            if (!DdfListItem(hDdf, Definition[i].Term,
                             Definition[i].Desc))
            {
                return (MRESULT)FALSE;
            }
        }

        /* terminate definition list */
        if (!DdfEndList(hDdf))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
}
```

DdfListItem - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfMetafile

DdfMetafile - Syntax

This function places a reference to a metafile into the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;      /* Handle to DDF returned by DdfInitialize. */
HMF       hmf;       /* The handle of the metafile to display. */
PRECTL    prclRect;  /* Size of the rectangle. */
BOOL      rc;        /* Success indicator. */

rc = DdfMetafile(hddf, hmf, prclRect);
```

DdfMetafile Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfMetafile Parameter - hmf

hmf (HMF) - input
The handle of the metafile to display.

DdfMetafile Parameter - prclRect

prclRect (PRECTL) - input

Size of the rectangle.

If not NULL, contains the size of the rectangle in which the metafile will be displayed. The aspect ratio of the metafile is adjusted to fit this rectangle.

DdfMetafile Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfMetafile - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

hmf (HMF) - input
The handle of the metafile to display.

prclRect (PRECTL) - input
Size of the rectangle.

If not NULL, contains the size of the rectangle in which the metafile will be displayed. The aspect ratio of the metafile is adjusted to fit this rectangle.

rc (BOOL) - returns
Success indicator.

TRUE

Successful completion

FALSE

Error occurred.

DdfMetafile - Remarks

There is a 3-byte ESC code overhead in the DDF internal buffer for this function. There is also a (MetaFilename length) overhead. Finally, the *prclRect* variable requires an additional 16 bytes of overhead in the DDF internal buffer.

DdfMetafile - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfMetafile - Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfMetafile - Example Code

After initializing a DDF buffer with [DdfInitialize](#), and loading a metafile with [GpiLoadMetaFile](#), the example uses [DdfMetafile](#) to place a reference to the metafile in the DDF buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#define INCL_GPIMETAFILES     /* MetaFiles */
#include <os2.h>
#include <pmhelp.h>

#define MF_HWND    0
#define ACVP_HAB   4

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HAB     hab;
    HWND    hwndInstance; /* help instance window */
    Hddf    hDdf;          /* DDF handle */
    HMF     hwndMetaFile;  /* metafile handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,           /* Default buffer size */
                0L,           /* Default increment */
            );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* get hab for this window */
            if ((hab = (HAB)WinQueryWindowULong(hwnd, ACVP_HAB)) == NULLHANDLE)
            {
                return (MRESULT)FALSE;
            }
    }
}
```

```

/* Load the Metafile to display */
if ((hwndMetaFile = GpiLoadMetaFile(hab, "SAMP.MET")) == NULLHANDLE)
{
    return (MRESULT)FALSE;
}

/* Save MetaFile hwnd in reserved word */
WinSetWindowULong(hwnd, MF_HWND, hwndMetaFile);

if (!DdfMetafile(hDdf, hwndMetaFile, NULL))
{
    return (MRESULT)FALSE;
}

return (hDdf);

case WM_CLOSE:
    GpiDeleteMetaFile((HMF)WinQueryWindowULong(hwnd, MF_HWND));
    WinDestroyWindow(WinQueryWindow(hwnd, QW_PARENT));

    return (MRESULT)TRUE;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfMetafile - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfPara

DdfPara - Syntax

This function creates a paragraph within the DDF buffer. It corresponds to the :p. tag. This function places a reference to a bit map in the DDF buffer.

```

#define INCL_DDF
#include <os2.h>

HDDF    hddf; /* Handle to DDF returned by DdfInitialize. */
BOOL    rc;    /* Success indicator. */

rc = DdfPara(hddf);

```

DdfPara Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfPara Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfPara - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfPara - Remarks

There is a 1-byte ESC code overhead in the DDF internal buffer for this function.

DdfPara - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfPara - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfPara - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf     hDdf;           /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance,    /* Handle of help instance */
                0L,              /* Default buffer size */
                0L,              /* Default increment */
            );

            if ( hDdf == NULLHANDLE )    /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* create paragraph in DDF buffer */
            if( !DdfPara( hDdf ) )
            {
                return (MRESULT)FALSE;
            }

            /* Change to large (100 x 100 dimensions) Courier font */
            if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
            {
                return (MRESULT)FALSE;
            }

            /* make the font BOLDFACE */
            if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
            {
                return (MRESULT)FALSE;
            }
    }
}
```

```

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGGRAY, CLR_BLUE ) )
        {
            return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if ( !DdfText( hDdf, "Sample Text" ) )
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfPara - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetColor

DdfSetColor - Syntax

This function sets the background and foreground colors of the displayed text.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
COLOR     fBackColor;    /* Specifies the desired background color. */
COLOR     fForeColor;    /* Specifies the desired foreground color. */
BOOL      rc;            /* Success indicator. */

rc = DdfSetColor(hddf, fBackColor, fForeColor);

```

DdfSetColor Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetColor Parameter - fBackColor

fBackColor (COLOR) - input
Specifies the desired background color.

DdfSetColor Parameter - fForeColor

fForeColor (COLOR) - input
Specifies the desired foreground color.

The following color value constants may be used for the foreground and background colors:

CLR_DEFAULT - used to set IPF default text color
CLR_BLACK
CLR_BLUE
CLR_RED
CLR_PINK
CLR_GREEN
CLR_CYAN
CLR_YELLOW
CLR_BROWN
CLR_DARKGRAY
CLR_DARKBLUE
CLR_DARKRED
CLR_DARKPINK
CLR_DARKGREEN
CLR_DARKCYAN
CLR_PALEGRAY

DdfSetColor Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetColor - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

fBackColor (COLOR) - input
Specifies the desired background color.

fForeColor (COLOR) - input
Specifies the desired foreground color.

The following color value constants may be used for the foreground and background colors:

CLR_DEFAULT - used to set IPF default text color
CLR_BLACK
CLR_BLUE
CLR_RED
CLR_PINK
CLR_GREEN
CLR_CYAN
CLR_YELLOW
CLR_BROWN
CLR_DARKGRAY
CLR_DARKBLUE
CLR_DARKRED
CLR_DARKPINK
CLR_DARKGREEN
CLR_DARKCYAN
CLR_PALEGRAY

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetColor - Remarks

There is a 4-byte ESC code overhead in the DDF internal buffer for the foreground color, and a 4-byte overhead for the background color, with this function.

DdfSetColor - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_BACKCOLOR (0x3003)
The background color is not valid.

HMERR_DDF_FORECOLOR (0x3004)
The foreground color is not valid.

DdfSetColor - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)

- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfSetColor - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance; /* help instance window */
    Hddf    hDdf;          /* DDF handle */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance, /* Handle of help instance */
            0L,           /* Default buffer size */
            0L,           /* Default increment */
        );

        if ( hDdf == NULLHANDLE ) /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
            return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
            return (MRESULT)FALSE;
        }
    }
}
```

```

        /* Write data into the buffer */
        if (!DdfText(hDdf, "Sample Text"))
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetColor - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetFont

DdfSetFont - Syntax

This function specifies a text font in the DDF buffer.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
PCSZ      pszFaceName;   /* Pointer to font name. */
ULONG     ulWidth;       /* Font width in points. */
ULONG     ulHeight;      /* Font height in points. */
BOOL      rc;            /* Success indicator. */

rc = DdfSetFont(hddf, pszFaceName, ulWidth,
               ulHeight);

```

DdfSetFont Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetFont Parameter - pszFaceName

pszFaceName (PCSZ) - input
Pointer to font name.

This parameter can be specified in two ways:

- An ASCIIZ string specifying the font name.
 - NULL or "default" to specify the default font.
-

DdfSetFont Parameter - ulWidth

ulWidth (ULONG) - input
Font width in points.

A point is approximately 1/72 of an inch.

DdfSetFont Parameter - ulHeight

ulHeight (ULONG) - input
Font height in points.

DdfSetFont Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetFont - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

pszFaceName (PCSZ) - input
Pointer to font name.

This parameter can be specified in two ways:

- An ASCIIZ string specifying the font name.
- NULL or "default" to specify the default font.

ulWidth (ULONG) - input
Font width in points.

A point is approximately 1/72 of an inch.

ulHeight (ULONG) - input
Font height in points.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetFont - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfSetFont - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfSetFont - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF               /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
```

```

{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf    hDdf;            /* DDF handle */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                          MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance,    /* Handle of help instance */
            0L,              /* Default buffer size */
            0L               /* Default increment */
        );

        if (hDdf == NULLHANDLE)    /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
            return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the font BOLDFACE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
            return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if ( !DdfText( hDdf, "Sample Text" ) )
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetFont - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetFontStyle

DdfSetFontStyle - Syntax

This function specifies a text font style in the DDF buffer.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;      /* Handle to DDF returned by DdfInitialize. */
ULONG     fFontStyle; /* Font style flag. */
BOOL      rc;        /* Success indicator. */

rc = DdfSetFontStyle(hddf, fFontStyle);
```

DdfSetFontStyle Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetFontStyle Parameter - fFontStyle

fFontStyle (ULONG) - input
Font style flag.

A NULL value for this parameter will set the font-style back to the default. Any of the following values can be specified:

```
FM_SEL_ITALIC
FM_SEL_BOLD
FM_SEL_UNDERSCORE
```

These values can be "ORed" together to combine different font styles.

Note: There is a 4-byte ESC code overhead in the DDF internal buffer for *fFontStyle* .

DdfSetFontStyle Return Value - rc

rc (BOOL) - returns
Success indicator.

```
TRUE
    Successful completion.
```

FALSE

Error occurred.

DdfSetFontStyle - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

fFontStyle (ULONG) - input
Font style flag.

A NULL value for this parameter *will* set the font-style back to the default. Any of the following values can be specified:

FM_SEL_ITALIC
FM_SEL_BOLD
FM_SEL_UNDERSCORE

These values can be "ORed" together to combine different font styles.

Note: There is a 4-byte ESC code overhead in the DDF internal buffer for *fFontStyle* .

rc (BOOL) - returns
Success indicator.

TRUE
Successful completion.
FALSE
Error occurred.

DdfSetFontStyle - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

HMERR_DDF_FONTSTYLE (0x3005)
The font style is not valid.

DdfSetFontStyle - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)

DdfSetFontStyle - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pnhlp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf     hDdf;           /* DDF handle */

    switch( ulMsg )
    {
    case HM_QUERY_DDF_DATA:
        /* get the help instance */
        hwndParent = WinQueryWindow( hwnd, QW_PARENT );
        hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
        hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                         MPFROMSHORT( HMQW_INSTANCE ), NULL );

        /* Allocate 1K Buffer (default) */
        hDdf = DdfInitialize(
            hwndInstance,    /* Handle of help instance */
            0L,              /* Default buffer size */
            0L               /* Default increment */
        );

        if ( hDdf == NULLHANDLE )    /* Check return code */
        {
            return (MRESULT)FALSE;
        }

        /* create paragraph in DDF buffer */
        if( !DdfPara( hDdf ) )
        {
            return (MRESULT)FALSE;
        }

        /* Change to large (100 x 100 dimensions) Courier font */
        if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the font BOLDFAE */
        if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
        {
            return (MRESULT)FALSE;
        }

        /* make the text display as BLUE on a PALE GRAY background */
        if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
        {
            return (MRESULT)FALSE;
        }

        /* Write data into the buffer */
        if ( !DdfText( hDdf, "Sample Text" ) )
        {
            return (MRESULT)FALSE;
        }

        return (MRESULT)hDdf;
    }
    return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}
```

DdfSetFontStyle - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetFontFormat

DdfSetFontFormat - Syntax

This function is used to turn formatting off or on. It corresponds to the **:lines.** tag.

```
#define INCL_DDF
#include <os2.h>

HDDF      hddf;          /* Handle to DDF returned by DdfInitialize. */
ULONG     fFormatType;    /* Formatting-activation flag. */
BOOL      rc;             /* Success indicator. */

rc = DdfSetFontFormat(hddf, fFormatType);
```

DdfSetFontFormat Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetFontFormat Parameter - fFormatType

fFormatType (ULONG) - input
Formatting-activation flag.

Only the following constants may be used in this parameter:

TRUE

FALSE	Turn formatting on.
	Turn formatting off.

DdfSetFormat Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetFormat - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

fFormatType (ULONG) - input
Formatting-activation flag.

Only the following constants may be used in this parameter:

TRUE	Turn formatting on.
FALSE	Turn formatting off.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion
FALSE	Error occurred.

DdfSetFormat - Remarks

If formatting is ON, there is a 3-byte ESC code overhead in the DDF internal buffer for this function. Otherwise, there is a 4-byte ESC code overhead.

DdfSetFormat - Errors

Possible returns from WinGetLastError

HMERR_DDF_MEMORY (0x3001)
Not enough memory is available.

DdfSetFormat - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfSetFormat - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfSetTextAlign](#) to specify left justified text in the DDF buffer when formatting is OFF. The example then uses [DdfSetFormat](#) to turn off formatting for text in the DDF buffer (corresponds to the IPF lines tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_GPIPRIMITIVES    /* Drawing Primitives/Attributes */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf    hDdf;            /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance, /* Handle of help instance */
                0L,          /* Default buffer size */
                0L,          /* Default increment */
            );

            if (hDdf == NULLHANDLE) /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* left justify text when formatting is OFF */
            if (!DdfSetTextAlign(hDdf, TA_LEFT))
            {
                return (MRESULT)FALSE;
            }

            /* turn formatting OFF */
            if (!DdfSetFormat(hDdf, FALSE))
            {

```



```

        return (MRESULT)FALSE;
    }

    if (!DdfText(hDdf,
        "Format OFF: This text should be Left Aligned!\n"))
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetFormat - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfSetTextAlign

DdfSetTextAlign - Syntax

This function defines whether left, center, or right text justification is to be used when text formatting is off.

```

#define INCL_DDF
#include <os2.h>

HDDF      hddf;    /* Handle to DDF returned by DdfInitialize. */
ULONG     fAlign;  /* Justification flag. */
BOOL      rc;      /* Success indicator. */

rc = DdfSetTextAlign(hddf, fAlign);

```

DdfSetTextAlign Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfSetTextAlign Parameter - fAlign

fAlign (ULONG) - input
Justification flag.

Only the following constants may be used:

TA_LEFT	Left-justify text.
TA_RIGHT	Right-justify text.
TA_CENTER	Center text.

DdfSetTextAlign Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfSetTextAlign - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

fAlign (ULONG) - input
Justification flag.

Only the following constants may be used:

TA_LEFT	Left-justify text.
TA_RIGHT	Right-justify text.
TA_CENTER	Center text.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfSetTextAlign - Remarks

It should be called before `DdfSetFormat` is called to turn off text formatting, and should not be called again until formatting is turned back on. Note that leading and trailing spaces are not stripped from the text as a result of this alignment.

DdfSetTextAlign - Errors

Possible returns from `WinGetLastError`

`HMERR_DDF_ALIGN_TYPE` (0x3002)
The alignment type is not valid.

DdfSetTextAlign - Related Functions

- [DdfBeginList](#)
 - [DdfBitmap](#)
 - [DdfEndList](#)
 - [DdfHyperText](#)
 - [DdfInform](#)
 - [DdfInitialize](#)
 - [DdfListItem](#)
 - [DdfMetafile](#)
 - [DdfPara](#)
 - [DdfSetColor](#)
 - [DdfSetFont](#)
 - [DdfSetFontStyle](#)
 - [DdfSetFormat](#)
 - [DdfSetTextAlign](#)
 - [DdfText](#)
-

DdfSetTextAlign - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses `DdfSetTextAlign` to specify left justified text in the DDF buffer when formatting is OFF. The example then uses [DdfSetFormat](#) to turn off formatting for text in the DDF buffer (corresponds to the IPF lines tag). For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_GPIPRIMITIVES    /* Drawing Primitives/Attributes*/
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf     hDdf;           /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
```

```

        hwndInstance, /* Handle of help instance */
        0L,           /* Default buffer size */
        0L           /* Default increment */
    );

    if (hDdf == NULLHANDLE) /* Check return code */
    {
        return (MRESULT)FALSE;
    }

    /* left justify text when formatting is OFF */
    if (!DdfSetTextAlign(hDdf, TA_LEFT))
    {
        return (MRESULT)FALSE;
    }

    /* turn formatting OFF */
    if (!DdfSetFormat(hDdf, FALSE))
    {
        return (MRESULT)FALSE;
    }

    if (!DdfText(hDdf,
        "Format OFF: This text should be Left Aligned!\n"))
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfSetTextAlign - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Errors](#)
[Remarks](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

DdfText

DdfText - Syntax

This function adds text to the DDF buffer.

```

#define INCL_DDF
#include <os2.h>

HDDF    hddf; /* Handle to DDF returned by DdfInitialize. */
PCSZ    pszText; /* Pointer to the text buffer to be formatted. */
BOOL    rc; /* Success indicator. */

```

```
rc = DdfText(hddf, pszText);
```

DdfText Parameter - hddf

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

DdfText Parameter - pszText

pszText (PCSZ) - input
Pointer to the text buffer to be formatted.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character.

DdfText Return Value - rc

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfText - Parameters

hddf (HDDF) - input
Handle to DDF returned by [DdfInitialize](#).

pszText (PCSZ) - input
Pointer to the text buffer to be formatted.

Note: There is a 3-byte ESC code overhead in the DDF internal buffer for each word in the text buffer. There is a 1-byte ESC code overhead for each blank and for each newline character.

rc (BOOL) - returns
Success indicator.

TRUE	Successful completion.
FALSE	Error occurred.

DdfText - Related Functions

- [DdfBeginList](#)
- [DdfBitmap](#)
- [DdfEndList](#)
- [DdfHyperText](#)
- [DdfInform](#)
- [DdfInitialize](#)
- [DdfListItem](#)
- [DdfMetafile](#)
- [DdfPara](#)
- [DdfSetColor](#)
- [DdfSetFont](#)
- [DdfSetFontStyle](#)
- [DdfSetFormat](#)
- [DdfSetTextAlign](#)
- [DdfText](#)

DdfText - Example Code

After initializing a DDF buffer with [DdfInitialize](#), the example uses [DdfPara](#) to start a new paragraph, [DdfSetFont](#) and [DdfSetFontStyle](#) to have the text displayed in a large, bold Courier font, [DdfSetColor](#) to change the text color, and [DdfText](#) to place text in the buffer. For a more detailed example and discussion of initializing DDF, see the [DdfInitialize](#) sample.

```
#define INCL_WINWINDOWMGR      /* General window management */
#define INCL_WINMESSAGEMGR    /* Message management */
#define INCL_DDF              /* Dynamic Data Facility */
#include <os2.h>
#include <pmhelp.h>

MRESULT WindowProc( HWND hwnd, ULONG ulMsg, MPARAM mp1, MPARAM mp2 )
{
    HWND    hwndParent;
    HWND    hwndInstance;    /* help instance window */
    Hddf     hDdf;           /* DDF handle */

    switch( ulMsg )
    {
        case HM_QUERY_DDF_DATA:
            /* get the help instance */
            hwndParent = WinQueryWindow( hwnd, QW_PARENT );
            hwndParent = WinQueryWindow( hwndParent, QW_PARENT );
            hwndInstance = (HWND)WinSendMsg( hwndParent, HM_QUERY,
                                             MPFROMSHORT( HMQW_INSTANCE ), NULL );

            /* Allocate 1K Buffer (default) */
            hDdf = DdfInitialize(
                hwndInstance,    /* Handle of help instance */
                0L,             /* Default buffer size */
                0L,             /* Default increment */
            );

            if (hDdf == NULLHANDLE)    /* Check return code */
            {
                return (MRESULT)FALSE;
            }

            /* create paragraph in DDF buffer */
            if( !DdfPara( hDdf ) )
            {
                return (MRESULT)FALSE;
            }

            /* Change to large (100 x 100 dimensions) Courier font */
            if( !DdfSetFont( hDdf, "Courier", 100L, 100L ) )
            {
                return (MRESULT)FALSE;
            }
    }
}
```

```

    }

    /* make the font BOLDFACE */
    if( !DdfSetFontStyle( hDdf, FM_SEL_BOLD ) )
    {
        return (MRESULT)FALSE;
    }

    /* make the text display as BLUE on a PALE GRAY background */
    if( !DdfSetColor( hDdf, CLR_PALEGRAY, CLR_BLUE ) )
    {
        return (MRESULT)FALSE;
    }

    /* Write data into the buffer */
    if ( !DdfText( hDdf, "Sample Text" ) )
    {
        return (MRESULT)FALSE;
    }

    return (MRESULT)hDdf;
}
return WinDefWindowProc( hwnd, ulMsg, mp1, mp2 );
}

```

DdfText - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Example Code](#)
[Related Functions](#)
[Glossary](#)

Help Manager Messages

The following is a summary of the messages sent by IPF and the application in response to user help requests.

HM_ACTIONBAR_COMMAND

This message is sent by IPF and notifies the application that a user has selected a tailored menu bar item.

HM_CONTROL

This message is sent to the application or the communication object by IPF prior to the addition of a push button in the control area of a window.

HM_CREATE_HELP_TABLE

This message is sent by the application and informs IPF to use the new help table indicated by this address in memory.

HM_DISMISS_WINDOW

This message is sent by the application and informs IPF to remove the active help window.

HM_DISPLAY_HELP

This message is sent by the application and informs IPF to display a specific help window.

HM_ERROR

This message notifies the application of an error caused by user interaction.

HM_EXT_HELP

This message is sent by the application and informs IPF to display the extended help window for the active application window.

HM_EXT_HELP_UNDEFINED

This message is sent by IPF and notifies the application that an extended help window has not been defined.

HM_GENERAL_HELP

This message is sent by the application and informs IPF to display the general help window for the active application window.

HM_GENERAL_HELP_UNDEFINED

This message is sent by IPF and notifies the application that a general help window has not been defined.

HM_HELP_CONTENTS

This message is sent by the application and informs IPF to display the Contents window.

HM_HELP_INDEX

This message is sent by the application and informs IPF to display the help index window.

HM_HELPSUBITEM_NOT_FOUND

This message is sent by IPF and notifies the application that a user has requested help on a field but that IPF cannot find a related entry in the help subtable.

HM_INFORM

This message is sent by IPF and notifies the application that a user has selected a hypertext field that was specified with the **reftype=inform** attribute of the **:link.** tag.

HM_INVALIDATE_DDF_DATA

This message is sent by the application and informs IPF that previous dynamic data formatting (DDF) information is no longer valid.

HM_KEYS_HELP

This message is sent by the application and informs IPF to display the keys help window.

HM_LOAD_HELP_TABLE

This message is sent by the application and provides IPF with the module handle that contains the help table, the help subtable, and the identity of the help table.

HM_NOTIFY

This message is sent by IPF and notifies the application or communication object that an event has occurred that the application may be interested in controlling.

HM_QUERY

This message is sent by the application and notifies IPF that the application requires IPF-specific information.

HM_QUERY_DDF_DATA

This message is sent by IPF and notifies the communication object that IPF has encountered the dynamic data formatting tag (**:ddf.**).

HM_QUERY_KEYS_HELP

This message is sent by IPF and notifies the application that a user has requested keys help for a function.

HM_REPLACE_HELP_FOR_HELP

This message is sent by the application and informs IPF to display the application-defined Help for Help window instead of the IPF Help for Help window.

HM_REPLACE_USING_HELP

This message is sent by the application and informs IPF to display the application-defined Using help window instead of the IPF Using help window.

HM_SET_ACTIVE_WINDOW

This message is sent by the application and enables the application to change the active application window with which the IPF help window is associated.

HM_SET_COVERPAGE_SIZE

This message is sent by the application and informs IPF to set the size of the coverpage window (the window within which all other IPF windows are displayed).

HM_SET_HELP_LIBRARY_NAME

This message is sent by the application and informs IPF to replace the list of help libraries specified in the initialization structure with a new list.

HM_SET_HELP_WINDOW_TITLE

This message is sent by the application and informs IPF to change the text of a help window title.

HM_SET_OBJCOM_WINDOW

This message is sent by the application and informs IPF to identify the communication object to which the HM_INFORM and HM_QUERY_DDF_DATA messages are sent.

HM_SET_SHOW_PANEL_ID

This message is sent by the application and informs IPF to display or hide window IDs for each help window.

HM_SET_USERDATA

This message is sent by the application and informs IPF to store data in the IPF data area.

HM_TUTORIAL

This message is sent by IPF and notifies the application when the user selects **Tutorial** choice from the Help menu bar.

HM_UPDATE_OBJCOM_WINDOW_CHAIN

This message is sent to the currently active communication object by the communication object who wants to withdraw from the communication chain.

A detailed description of the parameters and returns for these messages follows.

HM_ACTIONBAR_COMMAND

HM_ACTIONBAR_COMMAND - Syntax

This message is sent to the current active application window by the Help Manager to notify the application when the user selects a tailored action bar item.

```
param1
    USHORT    idCommand    /* Identity of the action bar item that was selected. */

param2
    ULONG     ulReserved   /* Reserved value, should be 0. */

returns
    ULONG     ulReserved   /* Reserved value, should be 0. */
```

HM_ACTIONBAR_COMMAND Field - idCommand

idCommand (USHORT)

Identity of the action bar item that was selected.

HM_ACTIONBAR_COMMAND Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_ACTIONBAR_COMMAND Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_ACTIONBAR_COMMAND - Parameters

idCommand (USHORT)
Identity of the action bar item that was selected.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_ACTIONBAR_COMMAND - Default Processing

None.

HM_ACTIONBAR_COMMAND - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

HM_CONTROL

HM_CONTROL - Syntax

This message is sent by the Help Manager to the child of the coverpage window to add a control in the control area of a window.

```
param1
    USHORT  usReserve  /* Reserved value. */
    USHORT  controlres /* Res number of the control that was selected. */

param2
    ULONG   ulReserved /* Reserved value. */

returns
    ULONG   ulReserved /* Reserved value, should be 0. */
```

HM_CONTROL Field - usReserve

usReserve (USHORT)
Reserved value.

HM_CONTROL Field - controlres

controlres (USHORT)
Res number of the control that was selected.

For author-defined push buttons, this is the res identification number that was specified with the push button tag (:pbutton.). For default push buttons, this is the res identification number defined in the PMHELP.H file.

HM_CONTROL Field - ulReserved

ulReserved (ULONG)
Reserved value.

HM_CONTROL Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_CONTROL - Parameters

usReserve (USHORT)
Reserved value.

controlres (USHORT)
Res number of the control that was selected.

For author-defined push buttons, this is the res identification number that was specified with the push button tag (:pbutton.). For default push buttons, this is the res identification number defined in the PMHELP.H file.

ulReserved (ULONG)
Reserved value.

ulReserved (ULONG)
Reserved value, should be 0.

HM_CONTROL - Remarks

If an application wants to filter any of the controls, it can subclass the child of the coverage window and intercept this message. If the application does not intercept this message, the Help Manager adds the control to the control area.

HM_CONTROL - Default Processing

None.

HM_CONTROL - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_CREATE_HELP_TABLE

HM_CREATE_HELP_TABLE - Syntax

This message is sent by the application to give the Help Manager a new help table.

```
param1
    PHELPTABLE  pHELPTABLE /* Help table. */

param2
    ULONG       ulReserved /* Reserved value, should be 0. */

returns
    ULONG       rc          /* Return code. */
```

HM_CREATE_HELP_TABLE Field - pHELPTABLE

pHELPTABLE (PHELPTABLE)

Help table.

This points to a help table structure; see HELPTABLE.

HM_CREATE_HELP_TABLE Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_CREATE_HELP_TABLE Return Value - rc

rc (ULONG)

Return code.

0

The procedure was successfully completed

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_CREATE_HELP_TABLE - Parameters

pHELPTABLE (PHELPTABLE)

Help table.

This points to a help table structure; see HELPTABLE.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The procedure was successfully completed

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_CREATE_HELP_TABLE - Default Processing

None.

HM_CREATE_HELP_TABLE - Topics

Select an item:

HM_DISMISS_WINDOW

HM_DISMISS_WINDOW - Syntax

This message tells the Help Manager to remove the active help window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    rc          /* Return code. */
```

HM_DISMISS_WINDOW Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_DISMISS_WINDOW Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_DISMISS_WINDOW Return Value - rc

rc (ULONG)
Return code.

0

The help window was successfully removed

Other

There was no associated help window.

See also the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_DISMISS_WINDOW - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The help window was successfully removed

Other

There was no associated help window.

See also the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_DISMISS_WINDOW - Remarks

If the user requests help from a primary or secondary window, and then interacts with the primary or secondary window without leaving help, the currently displayed help window might not be appropriate for the application window. This message gives the application the ability to remove that help window.

HM_DISMISS_WINDOW - Default Processing

None.

HM_DISMISS_WINDOW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_DISPLAY_HELP

HM_DISPLAY_HELP - Syntax

This message tells the Help Manager to display a specific help window.

```
param1
    USHORT    idHelpPanelId    /* Identity of the help window. */
    PSZ       pszHelpPanelName /* Name of the help window. */

param2
    USHORT    usTypeFlag       /* Flag indicating how to interpret the first parameter. */

returns
    ULONG     rc               /* Return code. */
```

HM_DISPLAY_HELP Field - idHelpPanelId

idHelpPanelId (USHORT)
Identity of the help window.

This points to a USHORT data type.

For a value of the *usTypeFlag* parameter of HM_PANELNAME.

HM_DISPLAY_HELP Field - pszHelpPanelName

pszHelpPanelName (PSZ)
Name of the help window.

This points to a string containing the name of the help window.

HM_DISPLAY_HELP Field - usTypeFlag

usTypeFlag (USHORT)
Flag indicating how to interpret the first parameter.

HM_RESOURCEID
Indicates the *param1* points to the identity of the help window.

HM_PANELNAME
Indicates the *param1* points to the name of the help window.

HM_DISPLAY_HELP Return Value - rc

rc (ULONG)
Return code.

0
The window was successfully displayed

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_DISPLAY_HELP - Parameters

idHelpPanelId (USHORT)
Identity of the help window.

This points to a USHORT data type.

For a value of the *usTypeFlag* parameter of HM_PANELNAME.

pszHelpPanelName (PSZ)
Name of the help window.

This points to a string containing the name of the help window.

usTypeFlag (USHORT)
Flag indicating how to interpret the first parameter.

HM_RESOURCEID
Indicates the *param1* points to the identity of the help window.

HM_PANELNAME
Indicates the *param1* points to the name of the help window.

rc (ULONG)
Return code.

0
The window was successfully displayed

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_DISPLAY_HELP - Remarks

param1 depends on the value of the *usTypeFlag* parameter.

HM_DISPLAY_HELP - Default Processing

None.

HM_DISPLAY_HELP - Topics

Select an item:

HM_ERROR

HM_ERROR - Syntax

This message notifies the application of an error caused by a user interaction.

```
param1
    ULONG    ulErrorCode    /* Error code. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */

returns
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

HM_ERROR Field - [ulErrorCode](#)

[ulErrorCode](#) (ULONG)
Error code.

A constant describing the type of error that occurred. The application can also receive some of these error constants in the *ulReserved* parameter of messages it has sent to the Help Manager.

The error constants are:

HMERR_LOAD_DLL
The resource DLL was unable to be loaded.

HMERR_NO_FRAME_WND_IN_CHAIN
There is no frame window in the window chain from which to find or set the associated help instance.

HMERR_INVALID_ASSOC_APP_WND
The application window handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_ASSOC_HELP_INST
The help instance handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_DESTROY_HELP_INST
The window handle specified as the help instance to destroy is not of the help instance class.

HMERR_NO_HELP_INST_IN_CHAIN
The parent or owner chain of the application window specified does not have an associated help instance.

HMERR_INVALID_HELP_INSTANCE_HDL
The handle specified to be a help instance does not have the class name of a Help Manager instance.

HMERR_INVALID_QUERY_APP_WND

The application window specified on a [WinQueryHelpInstance](#) function is not a valid window handle.

HMERR_HELP_INST_CALLED_INVALID

The handle of the instance specified on a call to the Help Manager does not have the class name of a Help Manager instance.

HMERR_HELPTABLE_UNDEFINE

The application did not provide a help table for context-sensitive help.

HMERR_HELP_INSTANCE_UNDEFINE

The help instance handle specified is invalid.

HMERR_HELPIITEM_NOT_FOUND

Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.

HMERR_INVALID_HELPSUBITEM_SIZE

The help subtable item size is less than 2.

HMERR_HELPSUBITEM_NOT_FOUND

Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.

HMERR_INDEX_NOT_FOUND

The index is not in the library file.

HMERR_CONTENT_NOT_FOUND

The library file does not have any content.

HMERR_OPEN_LIB_FILE

The library file cannot be opened.

HMERR_READ_LIB_FILE

The library file cannot be read.

HMERR_CLOSE_LIB_FILE

The library file cannot be closed.

HMERR_INVALID_LIB_FILE

Improper library file provided.

HMERR_NO_MEMORY

Unable to allocate the requested amount of memory.

HMERR_ALLOCATE_SEGMENT

Unable to allocate a segment of memory for memory allocation requests from the Help Manager.

HMERR_FREE_MEMORY

Unable to free allocated memory.

HMERR_PANEL_NOT_FOUND

Unable to find the requested help window.

HMERR_DATABASE_NOT_OPEN

Unable to read the unopened database.

HM_ERROR Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_ERROR Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_ERROR - Parameters

ulErrorCode (ULONG)
Error code.

A constant describing the type of error that occurred. The application can also receive some of these error constants in the *ulReserved* parameter of messages it has sent to the Help Manager.

The error constants are:

HMERR_LOAD_DLL

The resource DLL was unable to be loaded.

HMERR_NO_FRAME_WND_IN_CHAIN

There is no frame window in the window chain from which to find or set the associated help instance.

HMERR_INVALID_ASSOC_APP_WND

The application window handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_ASSOC_HELP_INST

The help instance handle specified on the [WinAssociateHelpInstance](#) function is not a valid window handle.

HMERR_INVALID_DESTROY_HELP_INST

The window handle specified as the help instance to destroy is not of the help instance class.

HMERR_NO_HELP_INST_IN_CHAIN

The parent or owner chain of the application window specified does not have an associated help instance.

HMERR_INVALID_HELP_INSTANCE_HDL

The handle specified to be a help instance does not have the class name of a Help Manager instance.

HMERR_INVALID_QUERY_APP_WND

The application window specified on a [WinQueryHelpInstance](#) function is not a valid window handle.

HMERR_HELP_INST_CALLED_INVALID

The handle of the instance specified on a call to the Help Manager does not have the class name of a Help Manager instance.

HMERR_HELPTABLE_UNDEFINE

The application did not provide a help table for context-sensitive help.

HMERR_HELP_INSTANCE_UNDEFINE

The help instance handle specified is invalid.

HMERR_HELPIITEM_NOT_FOUND

Context-sensitive help was requested but the ID of the main help item specified was not found in the help table.

HMERR_INVALID_HELPSUBITEM_SIZE

The help subtable item size is less than 2.

HMERR_HELPSUBITEM_NOT_FOUND

Context-sensitive help was requested but the ID of the help item specified was not found in the help subtable.

HMERR_INDEX_NOT_FOUND

The index is not in the library file.

HMERR_CONTENT_NOT_FOUND

The library file does not have any content.

HMERR_OPEN_LIB_FILE

The library file cannot be opened.

HMERR_READ_LIB_FILE

The library file cannot be read.

HMERR_CLOSE_LIB_FILE

The library file cannot be closed.

HMERR_INVALID_LIB_FILE
Improper library file provided.

HMERR_NO_MEMORY
Unable to allocate the requested amount of memory.

HMERR_ALLOCATE_SEGMENT
Unable to allocate a segment of memory for memory allocation requests from the Help Manager.

HMERR_FREE_MEMORY
Unable to free allocated memory.

HMERR_PANEL_NOT_FOUND
Unable to find the requested help window.

HMERR_DATABASE_NOT_OPEN
Unable to read the unopened database.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_ERROR - Remarks

There is no other way to communicate the error to the application since the user initiated communication, not the application. Other errors which occur when the application sends a message to the Help Manager are returned as the *ulReserved* parameter of the message.

The Help Manager does not display any error messages to the user. Instead, the Help Manager sends or returns all error notifications to the application so that it can display its own messages. This procedure ensures a consistent message interface for all user messages.

HM_ERROR - Default Processing

None.

HM_ERROR - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_EXT_HELP

HM_EXT_HELP - Syntax

When the Help Manager receives this message, it displays the extended help window for the active application panel.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    rc          /* Return code. */
```

HM_EXT_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP Return Value - rc

rc (ULONG)
Return code.

0	The extended help window was successfully displayed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_EXT_HELP - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

0
The extended help window was successfully displayed

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_EXT_HELP - Default Processing

None.

HM_EXT_HELP - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Default Processing](#)
[Glossary](#)

HM_EXT_HELP_UNDEFINED

HM_EXT_HELP_UNDEFINED - Syntax

This message is sent to the application by the Help Manager to notify it that an extended help window has not been defined.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    ulReserved /* Reserved value, should be 0. */
```

HM_EXT_HELP_UNDEFINED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_EXT_HELP_UNDEFINED - Remarks

When the extended help window is requested, the Help Manager searches the help table for its identity. If the extended help window identity associated with the current active window is zero, the Help Manager sends this message to the application to notify it that an extended help window has not been defined. The application then can:

- Ignore the request for help and not display a help window.
- Display its own window.
- Use the [HM_DISPLAY_HELP](#) message to tell the Help Manager to display a particular window.

HM_EXT_HELP_UNDEFINED - Default Processing

None.

HM_EXT_HELP_UNDEFINED - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_GENERAL_HELP

HM_GENERAL_HELP - Syntax

When the Help Manager receives this message, it displays the general help window for the active application window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    rc          /* Return code. */
```

HM_GENERAL_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_GENERAL_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_GENERAL_HELP Return Value - rc

rc (ULONG)

Return code.

0

The general help window was successfully displayed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_GENERAL_HELP - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The general help window was successfully displayed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_GENERAL_HELP - Default Processing

None.

HM_GENERAL_HELP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_GENERAL_HELP_UNDEFINED

HM_GENERAL_HELP_UNDEFINED - Syntax

This message is sent to the application by the Help Manager to notify it that a general help window has not been defined.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved. */

returns
    ULONG    ulReserved /* Reserved value, should be 0. */
```

HM_GENERAL_HELP_UNDEFINED Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_GENERAL_HELP_UNDEFINED Field - ulReserved

ulReserved (ULONG)
Reserved.

0	Reserved value, 0.
---	--------------------

HM_GENERAL_HELP_UNDEFINED Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_GENERAL_HELP_UNDEFINED - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved.

0	Reserved value, 0.
---	--------------------

ulReserved (ULONG)
Reserved value, should be 0.

HM_GENERAL_HELP_UNDEFINED - Remarks

When the general help window is requested, the Help Manager searches the help table for its identity. If the general help window identity associated with the current active window is zero, the Help Manager sends this message to the application to notify it that a general help window has not been defined. The application can then:

- Ignore the request for help and not display a help window.
- Display its own window.
- Use the [HM_DISPLAY_HELP](#) message to tell the Help Manager to display a particular window.

HM_GENERAL_HELP_UNDEFINED - Default Processing

None.

HM_GENERAL_HELP_UNDEFINED - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_HELP_CONTENTS

HM_HELP_CONTENTS - Syntax

When the Help Manager receives this message, it displays the help contents window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    rc          /* Return code. */
```

HM_HELP_CONTENTS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_HELP_CONTENTS Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_HELP_CONTENTS Return Value - rc

rc (ULONG)
Return code.

0	The help contents window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_HELP_CONTENTS - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

0	The help contents window was successfully displayed.
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_HELP_CONTENTS - Default Processing

None.

HM_HELP_CONTENTS - Topics

Select an item:

HM_HELP_INDEX

HM_HELP_INDEX - Syntax

When the Help Manager receives this message, it displays the help index window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    rc          /* Return code. */
```

HM_HELP_INDEX Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_HELP_INDEX Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_HELP_INDEX Return Value - rc

rc (ULONG)
Return code.

0

The help index window was successfully displayed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_HELP_INDEX - Parameters

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The help index window was successfully displayed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_HELP_INDEX - Default Processing

None.

HM_HELP_INDEX - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_HELPSUBITEM_NOT_FOUND

HM_HELPSUBITEM_NOT_FOUND - Syntax

The Help Manager sends this message to the application when the user requests help on a field and it cannot find a related entry in the help subtable.

```
param1
    USHORT usContext /* Type of window on which help was requested. */
```

```

param2
    SHORT    sTopic    /* Topic identifier. */
    SHORT    sSubTopic /* Subtopic identifier. */

returns
    BOOL     rc         /* Action indicator. */

```

HM_HELPSUBITEM_NOT_FOUND Field - usContext

usContext (USHORT)

Type of window on which help was requested.

HLPM_WINDOW	An application window
HLPM_FRAME	A frame window
HLPM_MENU	A menu window.

HM_HELPSUBITEM_NOT_FOUND Field - sTopic

sTopic (SHORT)

Topic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

window	Identity of the window containing the field on which help was requested.
menu	Identity of the submenu containing the field on which help was requested.

HM_HELPSUBITEM_NOT_FOUND Field - sSubTopic

sSubTopic (SHORT)

Subtopic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

control	Control identity of the cursored field and on which help was requested.
---------	---

For a value of the *usContext* parameter of HLPM_MENU:

-1	No menu item was selected
other	Menu item identity of the currently selected submenu item on which help was requested.

HM_HELPSUBITEM_NOT_FOUND Return Value - rc

rc (BOOL)

Action indicator.

HM_HELPSUBITEM_NOT_FOUND - Parameters

usContext (USHORT)

Type of window on which help was requested.

HLPM_WINDOW	An application window
HLPM_FRAME	A frame window
HLPM_MENU	A menu window.

sTopic (SHORT)

Topic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

window	Identity of the window containing the field on which help was requested.
menu	Identity of the submenu containing the field on which help was requested.

sSubTopic (SHORT)

Subtopic identifier.

For a value of the *usContext* parameter of HLPM_WINDOW or HLPM_FRAME:

control	Control identity of the cursored field and on which help was requested.
---------	---

For a value of the *usContext* parameter of HLPM_MENU:

-1	No menu item was selected
other	Menu item identity of the currently selected submenu item on which help was requested.

rc (BOOL)

Action indicator.

HM_HELPSUBITEM_NOT_FOUND - Remarks

If FALSE is returned from this message, the Help Manager displays the extended help window.

The application has the following options:

- Ignore the notification and not display help for that field or window.
- Display its own window.
- Use the [HM_DISPLAY_HELP](#) message to tell the Help Manager to display a particular window.

HM_HELPSUBITEM_NOT_FOUND - Default Processing

None.

HM_HELPSUBITEM_NOT_FOUND - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_INFORM

HM_INFORM - Syntax

This message is used by the Help Manager to notify the application when the user selects a hypertext field that was specified with the **reftype=inform** attribute of the **:link.** tag.

```
param1
    USHORT    idnum        /* Window identity. */

param2
    ULONG     ulReserved   /* Reserved value, should be 0. */

returns
    ULONG     ulReserved   /* Reserved value, should be 0. */
```

HM_INFORM Field - idnum

idnum (USHORT)
Window identity.

The identity that is associated with the hypertext field.

HM_INFORM Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_INFORM Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

0

Reserved value, zero.

HM_INFORM - Parameters

idnum (USHORT)

Window identity.

The identity that is associated with the hypertext field.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

0

Reserved value, zero.

HM_INFORM - Default Processing

None.

HM_INFORM - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_INVALIDATE_DDF_DATA

HM_INVALIDATE_DDF_DATA - Syntax

The application sends this message to IPF to indicate that the previous DDF data is no longer valid.

```

param1
    ULONG    rescount /* The count of DDFs to be invalidated. */

param2
    PUSHORT  resarray /* Pointer to an array. */

returns
    ULONG    rc       /* Return code. */

```

HM_INVALIDATE_DDF_DATA Field - rescount

rescount (ULONG)
The count of DDFs to be invalidated.

HM_INVALIDATE_DDF_DATA Field - resarray

resarray (PUSHORT)
Pointer to an array.

The pointer to an array of unsigned 16-bit (USHORT) integers that are the *res* numbers of DDFs to be invalidated.

Note: If both param1 and param2 are NULL, then all the DDFs in that page will be invalidated.

HM_INVALIDATE_DDF_DATA Return Value - rc

rc (ULONG)
Return code.

0	The procedure was successfully completed.
Other	See the values of the <i>errorcode</i> parameter of the HM_ERROR message.

HM_INVALIDATE_DDF_DATA - Parameters

rescount (ULONG)
The count of DDFs to be invalidated.

resarray (PUSHORT)
Pointer to an array.

The pointer to an array of unsigned 16-bit (USHORT) integers that are the *res* numbers of DDFs to be invalidated.

Note: If both param1 and param2 are NULL, then all the DDFs in that page will be invalidated.

rc (ULONG)
Return code.

0
The procedure was successfully completed.

Other
See the values of the *errorcode* parameter of the HM_ERROR message.

HM_INVALIDATE_DDF_DATA - Remarks

When IPF receives this message, it discards the current DDF data and sends a new HM_QUERY_DDF_DATA message to the object communication window.

This message should be sent to the child of the coverpage window handle.

HM_INVALIDATE_DDF_DATA - Default Processing

None.

HM_INVALIDATE_DDF_DATA - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_KEYS_HELP

HM_KEYS_HELP - Syntax

This message is sent by the application and informs the help manager to display the keys help window.

```
param1
    ULONG    ulReserved /* Reserved value, should be 0. */

param2
    ULONG    ulReserved /* Reserved value, should be 0. */

returns
    ULONG    rc          /* Return code. */
```

HM_KEYS_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_KEYS_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_KEYS_HELP Return Value - rc

rc (ULONG)
Return code.

0	The keys help window was successfully displayed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_KEYS_HELP - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

rc (ULONG)
Return code.

0	The keys help window was successfully displayed
Other	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_KEYS_HELP - Remarks

When the Help Manager receives this message, it sends a [HM_QUERY_KEYS_HELP](#) message to the active application window. The active application window is the window that was specified when the last [HM_SET_ACTIVE_WINDOW](#) message was sent. If no

[HM_SET_ACTIVE_WINDOW](#) message was issued, then the active application window is the window specified in the [WinAssociateHelpInstance](#) call.

The application must return one of the following:

- The identity of a keys help window in the *usHelpPanel* parameter of the [HM_QUERY_KEYS_HELP](#) message.
- Zero, if no action is to be taken by the Help Manager for keys help.

HM_KEYS_HELP - Default Processing

None.

HM_KEYS_HELP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_LOAD_HELP_TABLE

HM_LOAD_HELP_TABLE - Syntax

The application sends this message to give the Help Manager the module handle that contains the help table, the help subtable, and the identity of the help table.

```
param1
    USHORT    idHelpTable    /* Identity of the help table. */
    USHORT    fsidentityflag /* Help table identity indicator. */

param2
    HMODULE    MODULE        /* Resource identity. */

returns
    ULONG      rc            /* Return code. */
```

HM_LOAD_HELP_TABLE Field - idHelpTable

idHelpTable (USHORT)
Identity of the help table.

HM_LOAD_HELP_TABLE Field - fsidentityflag

fsidentityflag (USHORT)
Help table identity indicator.

0xFFFF
Reserved value.

HM_LOAD_HELP_TABLE Field - MODULE

MODULE (HMODULE)
Resource identity.

Handle of the module that contains the help table and help subtable.

HM_LOAD_HELP_TABLE Return Value - rc

rc (ULONG)
Return code.

0
The procedure was successfully completed.

Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_LOAD_HELP_TABLE - Parameters

idHelpTable (USHORT)
Identity of the help table.

fsidentityflag (USHORT)
Help table identity indicator.

0xFFFF
Reserved value.

MODULE (HMODULE)
Resource identity.

Handle of the module that contains the help table and help subtable.

rc (ULONG)
Return code.

0

The procedure was successfully completed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_LOAD_HELP_TABLE - Default Processing

None.

HM_LOAD_HELP_TABLE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_NOTIFY

HM_NOTIFY - Syntax

This message is used by the application to sub-class and change the behavior or appearance of the help window.

```
param1
    USHORT    controlres /* Res number of the control that was selected. */
    USHORT    usReserve  /* Reserved value, should be 0. */
    USHORT    usevent    /* The type of event which has occurred. */

param2
    ULONG     ulhwnd     /* Window handle of relevant window. */

returns
    BOOL      rc         /* Return code. */
```

HM_NOTIFY Field - controlres

controlres (USHORT)

Res number of the control that was selected.

For author-defined push buttons, this is the res number that was specified with the push button tag (**:pbutton.**). For default push

buttons, this is the res number defined in the PMHELP.H file.

HM_NOTIFY Field - usReserve

usReserve (USHORT)

Reserved value, should be 0.

Reserved for events other than CONTROL_SELECTED and HELP_REQUESTED.

HM_NOTIFY Field - usevent

usevent (USHORT)

The type of event which has occurred.

CONTROL_SELECTED

A control was selected.

HELP_REQUESTED

Help was requested.

OPEN_COVERPAGE

The coverpage is displayed.

OPEN_PAGE

The child window of the coverpage is opened.

SWAP_PAGE

The child window of the coverpage is swapped.

OPEN_INDEX

The index window is displayed.

OPEN_TOC

The table of contents window is displayed.

OPEN_HISTORY

The history window is displayed.

OPEN_LIBRARY

The new library is opened.

OPEN_SEARCH_HIT_LIST

The search list displayed.

HM_NOTIFY Field - ulhwnd

ulhwnd (ULONG)

Window handle of relevant window.

HM_NOTIFY Return Value - rc

rc (BOOL)

Return code.

TRUE

IPF will not format the controls and re-size the window.

FALSE

IPF will process as normal.

HM_NOTIFY - Parameters

controlres (USHORT)

Res number of the control that was selected.

For author-defined push buttons, this is the res number that was specified with the push button tag (**:pbutton.**). For default push buttons, this is the res number defined in the PMHELP.H file.

usReserve (USHORT)

Reserved value, should be 0.

Reserved for events other than CONTROL_SELECTED and HELP_REQUESTED.

usevent (USHORT)

The type of event which has occurred.

CONTROL_SELECTED	A control was selected.
HELP_REQUESTED	Help was requested.
OPEN_COVERPAGE	The coverpage is displayed.
OPEN_PAGE	The child window of the coverpage is opened.
SWAP_PAGE	The child window of the coverpage is swapped.
OPEN_INDEX	The index window is displayed.
OPEN_TOC	The table of contents window is displayed.
OPEN_HISTORY	The history window is displayed.
OPEN_LIBRARY	The new library is opened.
OPEN_SEARCH_HIT_LIST	The search list displayed.

ulhwnd (ULONG)

Window handle of relevant window.

rc (BOOL)

Return code.

TRUE	IPF will not format the controls and re-size the window.
FALSE	IPF will process as normal.

HM_NOTIFY - Remarks

This message is sent to the application to notify it of events that the application would be interested in controlling.

HM_NOTIFY - Default Processing

None.

HM_NOTIFY - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_QUERY

HM_QUERY - Syntax

This message is sent to IPF by the application to request IPF-specific information, such as the current Instance handle, the active communication object window, the active window, or the group number of the current window.

```
param1
    USHORT    usselectionid /* What is being requested. */
    USHORT    usmessageid  /* Type of window queried. */

param2
    PVOID     pvoid        /* Varies, depending on value selected above. */

returns
    ULONG     rc           /* Return code. */
```

HM_QUERY Field - usselectionid

usselectionid (USHORT)

What is being requested.

This parameter should be specified only if the query is for HMQW_VIEWPORT and should otherwise be coded as NULL.

Specifies whether a res ID, ID number, or group number is being requested. The value can be any of the following constants:

HMQVP_NUMBER

A pointer to a USHORT that holds the res ID of the window.

HMQVP_NAME

A pointer to a null-terminated string that holds the ID of the window.

HMQVP_GROUP

The group number of the window.

HM_QUERY Field - usmessageid

usmessageid (USHORT)

Type of window queried.

Specifies the type of window to query. The value can be any of the following constants:

HMQW_INDEX

The handle of the index window.

HMQW_TOC

The handle of the Table of Contents window.

HMQW_SEARCH

The handle of the Search Hitlist window.

HMQW_VIEWEDPAGES

The handle of the Viewed Pages window.

HMQW_LIBRARY

The handle of the Library List window.

HMQW_OBJCOM_WINDOW

The handle of the active communication window.

HMQW_INSTANCE

The handle of the help instance.

HMQW_COVERPAGE

The handle of the help manager multiple document interface (MDI) parent window. It is where the secondary windows are contained within the parent window.

HMQW_VIEWPORT

The handle of the viewport window specified in the low-order word of param1 and in param2.

When HMQW_VIEWPORT is specified in *usmessageid*, a value must be specified in *usselectionid* to indicate whether a res ID, ID number, or group number is being requested.

HMQW_GROUP_VIEWPORT

The group number of the window whose handle is specified in param2.

HMQW_RES_VIEWPORT

The res number of the window whose handle is specified in param2.

HMQW_ACTIVEVIEWPORT

The handle of the currently active window.

USERDATA

The previously stored user-data.

HM_QUERY Field - pvoid

pvoid (PVOID)

Varies, depending on value selected above.

param2 depends on the value of *param1 usmessageid*.

If *param1 usmessageid* is HMQW_VIEWPORT, then *param2* is a pointer to the res number, ID, or group ID.

If *param1 usmessageid* is HMQW_GROUP_VIEWPORT, then *param2* is the handle of the viewport for which the group number is assigned.

If *param1 usmessageid* is HMQW_RES_VIEWPORT, then *param2* is the handle of the viewport for which the res number is requested.

HM_QUERY Return Value - rc

rc (ULONG)

Return code.

0

The procedure was not successfully completed.

Other

The handle (HWND), group number (USHORT), or res number (USHORT) of the window, or the user data (USHORT), depending on the value of *param1 usselectionid* .

HM_QUERY - Parameters

usselectionid (USHORT)

What is being requested.

This parameter should be specified only if the query is for HMQW_VIEWPORT and should otherwise be coded as NULL.

Specifies whether a res ID, ID number, or group number is being requested. The value can be any of the following constants:

HMQVP_NUMBER

A pointer to a USHORT that holds the res ID of the window.

HMQVP_NAME

A pointer to a null-terminated string that holds the ID of the window.

HMQVP_GROUP

The group number of the window.

usmessageid (USHORT)

Type of window queried.

Specifies the type of window to query. The value can be any of the following constants:

HMQW_INDEX

The handle of the index window.

HMQW_TOC

The handle of the Table of Contents window.

HMQW_SEARCH

The handle of the Search Hitlist window.

HMQW_VIEWEDPAGES

The handle of the Viewed Pages window.

HMQW_LIBRARY

The handle of the Library List window.

HMQW_OBJCOM_WINDOW

The handle of the active communication window.

HMQW_INSTANCE

The handle of the help instance.

HMQW_COVERPAGE

The handle of the help manager multiple document interface (MDI) parent window. It is where the secondary windows are contained within the parent window.

HMQW_VIEWPORT

The handle of the viewport window specified in the low-order word of param1 and in param2.

When HMQW_VIEWPORT is specified in *usmessageid* , a value must be specified in *usselectionid* to indicate whether a res ID, ID number, or group number is being requested.

HMQW_GROUP_VIEWPORT

HMQW_RES_VIEWPORT

The group number of the window whose handle is specified in param2.

HMQW_ACTIVEVIEWPORT

The res number of the window whose handle is specified in param2.

USERDATA

The handle of the currently active window.

The previously stored user-data.

pvoid (PVOID)

Varies, depending on value selected above.

param2 depends on the value of *param1 usmessageid* .

If *param1 usmessageid* is HMQW_VIEWPORT, then *param2* is a pointer to the res number, ID, or group ID.

If *param1 usmessageid* is HMQW_GROUP_VIEWPORT, then *param2* is the handle of the viewport for which the group number is assigned.

If *param1 usmessageid* is HMQW_RES_VIEWPORT, then *param2* is the handle of the viewport for which the res number is requested.

rc (ULONG)

Return code.

0

The procedure was not successfully completed.

Other

The handle (HWND), group number (USHORT), or res number (USHORT) of the window, or the user data (USHORT), depending on the value of *param1 usselectionid* .

HM_QUERY - Default Processing

None.

HM_QUERY - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_QUERY_DDF_DATA

HM_QUERY_DDF_DATA - Syntax

This message is sent to the communication object window by IPF when it encounters the dynamic data formatting (:**ddf.**) tag.

```

param1
    HWND    pageclienthwnd /* Client handle. */

param2
    ULONG    resid          /* The res ID associated with the DDF tag. */

returns
    Hddf     rc              /* Return code. */

```

HM_QUERY_DDF_DATA Field - pageclienthwnd

pageclienthwnd (HWND)
Client handle.

The client handle of the page that contains the object communication window.

HM_QUERY_DDF_DATA Field - resid

resid (ULONG)
The res ID associated with the DDF tag.

HM_QUERY_DDF_DATA Return Value - rc

rc (Hddf)
Return code.

0
An error has occurred in the application's DDF processing.

Other
The DDF handle to be displayed.

Note: Once this handle has been returned, the Hddf handle can no longer be used by the application.

HM_QUERY_DDF_DATA - Parameters

pageclienthwnd (HWND)
Client handle.

The client handle of the page that contains the object communication window.

resid (ULONG)
The res ID associated with the DDF tag.

rc (HDDF)

Return code.

0

An error has occurred in the application's DDF processing.

Other

The DDF handle to be displayed.

Note: Once this handle has been returned, the HDDF handle can no longer be used by the application.

HM_QUERY_DDF_DATA - Remarks

Upon receiving this message, the communication object calls `DdfInitialize` to indicate the start of dynamic data formatting (DDF). Any combination of other DDF calls are then made to describe this data. When this is complete, the communication object finishes processing this message, indicating that the DDF data is complete. After that time, the DDF handle received from `DdfInitialize` is considered invalid.

HM_QUERY_DDF_DATA - Default Processing

None.

HM_QUERY_DDF_DATA - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_QUERY_KEYS_HELP

HM_QUERY_KEYS_HELP - Syntax

When the user requests the keys help function, the Help Manager sends this message to the application.

```
param1
    ULONG    ulReserved    /* Reserved value, should be 0. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */
```

```
returns
    USHORT  usHelpPanel /* Help panel ID. */
```

HM_QUERY_KEYS_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_QUERY_KEYS_HELP Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_QUERY_KEYS_HELP Return Value - usHelpPanel

usHelpPanel (USHORT)
Help panel ID.

The identity of the application-defined keys help window to be displayed.

0	Do nothing
Other	Identity of the keys help window to be displayed.

HM_QUERY_KEYS_HELP - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

usHelpPanel (USHORT)
Help panel ID.

The identity of the application-defined keys help window to be displayed.

0	Do nothing
Other	Identity of the keys help window to be displayed.

HM_QUERY_KEYS_HELP - Remarks

The application responds by returning the identity of the requested keys help window. The Help Manager then displays that help window. Returning 0 in the *usHelpPanel* parameter indicates that the Help Manager should do nothing for the keys help function.

HM_QUERY_KEYS_HELP - Default Processing

None.

HM_QUERY_KEYS_HELP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_REPLACE_HELP_FOR_HELP

HM_REPLACE_HELP_FOR_HELP - Syntax

This message tells the Help Manager to display the application-defined Help for Help window instead of the Help Manager Help for Help window.

```
param1
    USHORT    idHelpForHelpPanel    /* Identity of the application-defined Help for Help window. */

param2
    ULONG     ulReserved            /* Reserved value, should be 0. */

returns
    ULONG     ulReserved            /* Reserved value, should be 0. */
```

HM_REPLACE_HELP_FOR_HELP Field - idHelpForHelpPanel

idHelpForHelpPanel (USHORT)

Identity of the application-defined Help for Help window.

0

Use the Help Manager Help for Help window.

Other

Identity of the application-defined Help for Help window.

HM_REPLACE_HELP_FOR_HELP Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_REPLACE_HELP_FOR_HELP Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_REPLACE_HELP_FOR_HELP - Parameters

idHelpForHelpPanel (USHORT)

Identity of the application-defined Help for Help window.

0

Use the Help Manager Help for Help window.

Other

Identity of the application-defined Help for Help window.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

HM_REPLACE_HELP_FOR_HELP - Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information provided in the Help Manager Help for Help window.

HM_REPLACE_HELP_FOR_HELP - Default Processing

None.

HM_REPLACE_HELP_FOR_HELP - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_REPLACE_USING_HELP

HM_REPLACE_USING_HELP - Syntax

This message tells the Help Manager to display the application-defined Using help window instead of the Help Manager Using help window.

```
param1
    USHORT    idUsingHelpPanel

param2
    ULONG     ulReserved        /* Reserved value, should be 0. */

returns
    ULONG     ulReserved        /* Reserved value, should be 0. */
```

HM_REPLACE_USING_HELP Field - idUsingHelpPanel

idUsingHelpPanel (USHORT)

The identity of the application-defined Using Help window.

- | | |
|-------|--|
| 0 | Use the Help Manager Using Help window. |
| Other | The identity of the application-defined Using Help window. |
-

HM_REPLACE_USING_HELP Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_REPLACE_USING_HELP Return Value - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_REPLACE_USING_HELP - Parameters

idUsingHelpPanel (USHORT)

The identity of the application-defined Using Help window.

0

Use the Help Manager Using Help window.

Other

The identity of the application-defined Using Help window.

ulReserved (ULONG)

Reserved value, should be 0.

ulReserved (ULONG)

Reserved value, should be 0.

HM_REPLACE_USING_HELP - Remarks

An application may prefer to provide information that is more specific to itself, rather than the more general help information that is provided in the Help Manager Using help window. The guidelines that define the current CUA interface recommend the **Using help** choice be provided in a pull-down menu from the **Help** choice.

HM_REPLACE_USING_HELP - Default Processing

None.

HM_REPLACE_USING_HELP - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_SET_ACTIVE_WINDOW

HM_SET_ACTIVE_WINDOW - Syntax

This message allows the application to change the window with which the Help Manager communicates and the window to which the help window is to be positioned.

```
param1
    HWND    hwndActiveWindow    /* The handle of the window to be made active. */

param2
    HWND    hwndRelativeWindow /* The handle of the window next to which the help window is to be positioned. */

returns
    ULONG   rc                  /* Return code. */
```

HM_SET_ACTIVE_WINDOW Field - hwndActiveWindow

hwndActiveWindow (HWND)

The handle of the window to be made active.

Its window procedure receives all messages from the Help Manager until the application changes the active window with another HM_SET_ACTIVE_WINDOW message.

HM_SET_ACTIVE_WINDOW Field - hwndRelativeWindow

hwndRelativeWindow (HWND)

The handle of the window next to which the help window is to be positioned.

The handle of the application window next to which the Help Manager will position a new help window.

HWND_PARENT

This Help Manager defined constant tells the Help Manager to trace the parent chain of the window that had the focus when the user requested help.

Other

Handle of the window next to which the help window is to be positioned.

HM_SET_ACTIVE_WINDOW Return Value - rc

rc (ULONG)

Return code.

0

Other	The procedure has been successfully completed.
	See the values of the <i>ulErrorCode</i> parameter of the HM_ERROR message.

HM_SET_ACTIVE_WINDOW - Parameters

hwndActiveWindow (HWND)

The handle of the window to be made active.

Its window procedure receives all messages from the Help Manager until the application changes the active window with another HM_SET_ACTIVE_WINDOW message.

hwndRelativeWindow (HWND)

The handle of the window next to which the help window is to be positioned.

The handle of the application window next to which the Help Manager will position a new help window.

HWND_PARENT

This Help Manager defined constant tells the Help Manager to trace the parent chain of the window that had the focus when the user requested help.

Other

Handle of the window next to which the help window is to be positioned.

rc (ULONG)

Return code.

0

The procedure has been successfully completed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_ACTIVE_WINDOW - Remarks

Normally the Help Manager communicates with the application window with which the Help Manager instance has been associated. The help window is positioned next to this same application window.

If the *hwndActiveWindow* parameter is 0, the *hwndRelativeWindow* parameter is set to 0. That is, if the active window is NULL HANDLE, the relative window is not used.

HM_SET_ACTIVE_WINDOW - Default Processing

None.

HM_SET_ACTIVE_WINDOW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

HM_SET_COVERPAGE_SIZE

HM_SET_COVERPAGE_SIZE - Syntax

This message is sent to IPF by the application to set the size of the coverpage, the window within which all other IPF windows are displayed.

```
param1
    PRECTL    coverpagerectl /* Pointer to RECTL containing the size of the coverpage. */

param2
    ULONG     ulReserved    /* Reserved value, should be 0. */

returns
    ULONG     rc            /* Return code. */
```

HM_SET_COVERPAGE_SIZE Field - coverpagerectl

coverpagerectl (PRECTL)
Pointer to RECTL containing the size of the coverpage.

HM_SET_COVERPAGE_SIZE Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_SET_COVERPAGE_SIZE Return Value - rc

rc (ULONG)
Return code.

0
The procedure was successfully completed.

Other
See the values of the *errorcode* parameter of the HM_ERROR message.

HM_SET_COVERPAGE_SIZE - Parameters

coverpagerectl (PRECTL)

Pointer to RECTL containing the size of the coverpage.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The procedure was successfully completed.

Other

See the values of the *errorcode* parameter of the HM_ERROR message.

HM_SET_COVERPAGE_SIZE - Remarks

The default size for the coverpage of a book is the full width of the screen, while the default size for a help file is one-half the width of the screen.

This message takes effect immediately, changing the size of the coverpage. If the coverpage is not currently open, the requested size is saved for the next open.

HM_SET_COVERPAGE_SIZE - Default Processing

None.

HM_SET_COVERPAGE_SIZE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_SET_HELP_LIBRARY_NAME

HM_SET_HELP_LIBRARY_NAME - Syntax

This message identifies a list of help window library names to the Help Manager instance.

```
param1
    PSZ    pszHelpLibraryName /* Library name. */

param2
    ULONG  ulReserved         /* Reserved value, should be 0. */

returns
    ULONG  rc                 /* Return code. */
```

HM_SET_HELP_LIBRARY_NAME Field - pszHelpLibraryName

pszHelpLibraryName (PSZ)
Library name.

This points to a string that contains a list of help window library names that will be searched by the Help Manager for the requested help window. The names must be separated by a blank.

HM_SET_HELP_LIBRARY_NAME Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_SET_HELP_LIBRARY_NAME Return Value - rc

rc (ULONG)
Return code.

- 0
The newly specified library successfully replaced the current help window library name.
- Other
See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_LIBRARY_NAME - Parameters

pszHelpLibraryName (PSZ)
Library name.

This points to a string that contains a list of help window library names that will be searched by the Help Manager for the requested help window. The names must be separated by a blank.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The newly specified library successfully replaced the current help window library name.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_LIBRARY_NAME - Remarks

Any subsequent communication to the Help Manager with this message replaces the current list of names with the newly specified list.

When help is requested, the Help Manager will search each library in the list for the requested help window.

HM_SET_HELP_LIBRARY_NAME - Default Processing

None.

HM_SET_HELP_LIBRARY_NAME - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_SET_HELP_WINDOW_TITLE

HM_SET_HELP_WINDOW_TITLE - Syntax

This message allows the application to change the window text of a help window title.

```
param1
    PSZ      pszHelpWindowTitle /* Help window title. */

param2
    ULONG    ulReserved          /* Reserved value, should be 0. */

returns
```

ULONG **rc** /* Return code. */

HM_SET_HELP_WINDOW_TITLE Field - pszHelpWindowTitle

pszHelpWindowTitle (PSZ)

Help window title.

This points to a string containing the new Help Window title.

HM_SET_HELP_WINDOW_TITLE Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_SET_HELP_WINDOW_TITLE Return Value - rc

rc (ULONG)

Return code.

0

The window title was successfully set.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_WINDOW_TITLE - Parameters

pszHelpWindowTitle (PSZ)

Help window title.

This points to a string containing the new Help Window title.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The window title was successfully set.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_HELP_WINDOW_TITLE - Default Processing

None.

HM_SET_HELP_WINDOW_TITLE - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Default Processing](#)

[Glossary](#)

HM_SET_OBJCOM_WINDOW

HM_SET_OBJCOM_WINDOW - Syntax

This message is sent to IPF by the application to identify the communication object window to which the HM_INFORM and HM_QUERY_DDF_DATA messages will be sent. This message is not necessary if the communication object does not expect to receive either of these messages.

```
hwndparam1
    HWND    objcomhwnd    /* Handle of the communication object window to be set. */

param2
    ULONG    ulReserved    /* Reserved value, should be 0. */

returns
    HWND    hwndprevioushwnd /* The handle of the previous communication object window. */
```

HM_SET_OBJCOM_WINDOW Field - objcomhwnd

objcomhwnd (HWND)

Handle of the communication object window to be set.

HM_SET_OBJCOM_WINDOW Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_SET_OBJCOM_WINDOW Return Value - hwndprevioushwnd

hwndprevioushwnd (HWND)

The handle of the previous communication object window.

HM_SET_OBJCOM_WINDOW - Parameters

objcomhwnd (HWND)

Handle of the communication object window to be set.

ulReserved (ULONG)

Reserved value, should be 0.

hwndprevioushwnd (HWND)

The handle of the previous communication object window.

HM_SET_OBJCOM_WINDOW - Remarks

HM_INFORM and HM_QUERY_DDF_DATA messages which are not processed must be passed to the previous communication object window which was returned when HM_SET_OBJECT_WINDOW was sent.

HM_SET_OBJCOM_WINDOW - Default Processing

None.

HM_SET_OBJCOM_WINDOW - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Returns](#)

[Remarks](#)

[Default Processing](#)

[Glossary](#)

HM_SET_SHOW_PANEL_ID

HM_SET_SHOW_PANEL_ID - Syntax

This message tells the Help Manager to display, hide, or toggle the window identity for each help window displayed.

```
param1
    USHORT    fsShowPanelId /* The show window identity indicator. */

param2
    ULONG     ulReserved    /* Reserved value, should be 0. */

rc
    ULONG     rc            /* Return code. */
```

HM_SET_SHOW_PANEL_ID Field - fsShowPanelId

fsShowPanelId (USHORT)

The show window identity indicator.

CMIC_HIDE_PANEL_ID

Sets the show option off and the window identity is not displayed.

CMIC_SHOW_PANEL_ID

Sets the show option on and the window identity is displayed.

CMIC_TOGGLE_PANEL_ID

Toggles the display of the window identity.

HM_SET_SHOW_PANEL_ID Field - ulReserved

ulReserved (ULONG)

Reserved value, should be 0.

HM_SET_SHOW_PANEL_ID Parameter - rc

rc (ULONG)

Return code.

0

The show window identity indicator was successfully changed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_SHOW_PANEL_ID - Parameters

fsShowPanelId (USHORT)

The show window identity indicator.

CMIC_HIDE_PANEL_ID

Sets the show option off and the window identity is not displayed.

CMIC_SHOW_PANEL_ID

Sets the show option on and the window identity is displayed.

CMIC_TOGGLE_PANEL_ID

Toggles the display of the window identity.

ulReserved (ULONG)

Reserved value, should be 0.

rc (ULONG)

Return code.

0

The show window identity indicator was successfully changed.

Other

See the values of the *ulErrorCode* parameter of the [HM_ERROR](#) message.

HM_SET_SHOW_PANEL_ID - Default Processing

None.

HM_SET_SHOW_PANEL_ID - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Default Processing](#)

[Glossary](#)

HM_SET_USERDATA

HM_SET_USERDATA - Syntax

The application sends this message to IPF to store data in the IPF data area.

```
param1
    ULONG    ulReserved    /* Reserved value, should be 0. */

param2
    VOID     usrdata        /* 4-byte user data area. */
```

```
rc      ULONG      rc      /* Return code. */
```

HM_SET_USERDATA Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_SET_USERDATA Field - usrdata

usrdata (VOID)
4-byte user data area.

HM_SET_USERDATA Parameter - rc

rc (ULONG)
Return code.

TRUE	The user data was successfully stored.
FALSE	The call failed.

HM_SET_USERDATA - Parameters

ulReserved (ULONG)
Reserved value, should be 0.

usrdata (VOID)
4-byte user data area.

rc (ULONG)
Return code.

TRUE	The user data was successfully stored.
FALSE	The call failed.

HM_SET_USERDATA - Default Processing

None.

HM_SET_USERDATA - Topics

Select an item:

[Syntax](#)

[Parameters](#)

[Default Processing](#)

[Glossary](#)

HM_TUTORIAL

HM_TUTORIAL - Syntax

The Help Manager sends this message to the application window when the user selects the Tutorial choice from a help window.

```
param1
    PSZ      pszTutorialName /* Default tutorial name. */

param2
    ULONG    ulReserved      /* Reserved value, should be 0. */

returns
    ULONG    ulReserved      /* Reserved value, should be 0. */
```

HM_TUTORIAL Field - pszTutorialName

pszTutorialName (PSZ)
Default tutorial name.

This points to a string that contains the name of the default tutorial program specified in the Help Manager initialization structure. A tutorial name specified in the help window definition overrides this default tutorial program.

HM_TUTORIAL Field - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_TUTORIAL Return Value - ulReserved

ulReserved (ULONG)
Reserved value, should be 0.

HM_TUTORIAL - Parameters

pszTutorialName (PSZ)
Default tutorial name.

This points to a string that contains the name of the default tutorial program specified in the Help Manager initialization structure. A tutorial name specified in the help window definition overrides this default tutorial program.

ulReserved (ULONG)
Reserved value, should be 0.

ulReserved (ULONG)
Reserved value, should be 0.

HM_TUTORIAL - Remarks

The application then calls its own tutorial program.

HM_TUTORIAL - Default Processing

None.

HM_TUTORIAL - Topics

Select an item:
[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

HM_UPDATE_OBJCOM_WINDOW_CHAIN

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Syntax

This message is sent to the currently active communication object by the communication object who wants to withdraw from the communication chain.

```
param1
    HWND    hwnd    /* The handle of the object to be withdrawn from the communication chain. */

param2
    HWND    hwnd    /* Window containing the handle of the object to be replaced. */

returns
    ULONG    ulReserved /* Reserved value, should be 0. */
```

HM_UPDATE_OBJCOM_WINDOW_CHAIN Field - hwnd

hwnd (HWND)
The handle of the object to be withdrawn from the communication chain.

HM_UPDATE_OBJCOM_WINDOW_CHAIN Field - hwnd

hwnd (HWND)
Window containing the handle of the object to be replaced.

HM_UPDATE_OBJCOM_WINDOW_CHAIN Return Value - ulReser

ulReserved (ULONG)
Reserved value, should be 0.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Parameters

hwnd (HWND)
The handle of the object to be withdrawn from the communication chain.

hwnd (HWND)
Window containing the handle of the object to be replaced.

ulReserved (ULONG)
Reserved value, should be 0.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Remarks

The object that receives this message should check to see if the object handle returned from [HM_SET_OBJCOM_WINDOW](#) is equal to the handle in *param1* . If the handle is equal, then the handle in *param1* should be replaced by the handle in *param2* . If the handle is not equal *and* the handle previously received is not NULL HANDLE, then send HM_UPDATE_OBJCOM_WINDOW_CHAIN to that object.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Default Processing

None.

HM_UPDATE_OBJCOM_WINDOW_CHAIN - Topics

Select an item:

[Syntax](#)
[Parameters](#)
[Returns](#)
[Remarks](#)
[Default Processing](#)
[Glossary](#)

Notices

(November 1996)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM reseller or IBM marketing representative.

Copyright Notices

COPYRIGHT LICENSE: This publication contains printed sample application programs in source language, which illustrate OS/2 programming techniques. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the OS/2 application programming interface.

Each copy of any portion of these sample programs or any derivative work, which is distributed to others, must include a copyright notice as follows: "(C) (your company name) (year). All rights reserved."

(C) Copyright International Business Machines Corporation 1994, 1996. All rights reserved.

Note to U.S. Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Disclaimers

Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

- IBM Director of Licensing
- IBM Corporation
- 500 Columbus Avenue
- Thornwood, NY 10594
- U.S.A.

Asia-Pacific users can inquire, in writing, to the IBM Director of Intellectual Property and Licensing, IBM World Trade Asia Corporation, 2-31 Roppongi 3-chome, Minato-ku, Tokyo 106, Japan.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Department LZKS, 11400 Burnet Road, Austin, TX 78758 U.S.A. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Common User Access
CUA
IBM
Operating System/2
OS/2
Personal System/2
Presentation Manager
Systems Application Architecture

The following terms are trademarks of other companies:

Adobe	Adobe Systems Incorporated
Helvetica	Linotype AG
Intel486	Intel Corporation
PostScript	Adobe Systems Incorporated
Times New Roman	Monotype Corporation

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Glossary

This glossary defines many of the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing*, as well as

terms specific to the OS/2 operating system and the Presentation Manager. It is not a complete glossary for the entire OS/2 operating system; nor is it a complete dictionary of computer terms.

Other primary sources for these definitions are:

- The *American National Standard Dictionary for Information Systems* , ANSI X3.172-1990, copyrighted 1990 by the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. These definitions are identified by the symbol (A) after the definition.
- The *Information Technology Vocabulary* , developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary Listing

Select a starting letter of glossary terms:

A	N
B	O
C	P
D	Q
E	R
F	S
G	T
H	U
I	V
J	W
K	X
L	Y
M	Z

Glossary - A

accelerator - In SAA Common User Access architecture, a key or combination of keys that invokes an application-defined function.

accelerator table - A table used to define which key strokes are treated as *accelerators* and the commands they are translated into.

access mode - The manner in which an application gains access to a file it has opened. Examples of access modes are read-only, write-only, and read/write.

access permission - All access rights that a user has regarding an object. (I)

action - One of a set of defined tasks that a computer performs. Users request the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action name from an action bar or menu.

action bar - In SAA Common User Access architecture, the area at the top of a window that contains choices that give a user access to actions available in that window.

action point - The current position on the screen at which the pointer is pointing. Contrast with *hot spot* and *input focus* .

active program - A program currently running on the computer. An active program can be interactive (running and receiving input from the user) or noninteractive (running but not receiving input from the user). See also *interactive program* and *noninteractive program* .

active window - The window with which the user is currently interacting.

address space - (1) The range of addresses available to a program. (A) (2) The area of virtual storage available for a particular job.

alphanumeric video output - Output to the logical video buffer when the video adapter is in text mode and the logical video buffer is addressed by an application as a rectangular array of character cells.

American National Standard Code for Information Interchange - The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

Note: IBM has defined an extension to ASCII code (characters 128-255).

anchor - A window procedure that handles Presentation Manager message conversions between an icon procedure and an application.

anchor block - An area of Presentation-Manager-internal resources to allocated process or thread that calls WinInitialize.

anchor point - A point in a window used by a program designer or by a window manager to position a subsequently appearing window.

ANSI - American National Standards Institute.

APA - All points addressable.

API - Application programming interface.

application - A collection of software components used to perform specific types of work on a computer; for example, a payroll application, an airline reservation application, a network application.

application object - In SAA Advanced Common User Access architecture, a form that an application provides for a user; for example, a spreadsheet form. Contrast with *user object* .

application programming interface (API) - A functional interface supplied by the operating system or by a separately orderable licensed program that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

application-modal - Pertaining to a message box or dialog box for which processing must be completed before further interaction with any other window owned by the same application may take place.

area - In computer graphics, a filled shape such as a solid rectangle.

ASCII - American National Standard Code for Information Interchange.

ASCIIZ - A string of ASCII characters that is terminated with a byte containing the value 0.

aspect ratio - In computer graphics, the width-to-height ratio of an area, symbol, or shape.

asynchronous (ASYNC) - (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions. See also *synchronous* .

atom - A constant that represents a string. As soon as a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table* . See also *integer atom* .

atom table - A table used to relate *atoms* with the strings that they represent. Also in the table is the mechanism by which the presence of a string can be checked.

atomic operation - An operation that completes its work on an object before another operation can be performed on the same object.

attribute - A characteristic or property that can be controlled, usually to obtain a required appearance; for example, the color of a line. See also *graphics attributes* and *segment attributes* .

automatic link - In Information Presentation Facility (IPF), a link that begins a chain reaction at the primary window. When the user selects the primary window, an automatic link is activated to display secondary windows.

AVIO - Advanced Video Input/Output.

Glossary - B

Bézier curve - (1) A mathematical technique of specifying smooth continuous lines and surfaces, which require a starting point and a finishing point with several intermediate points that influence or control the path of the linking curve. Named after Dr. P. Bézier. (2) (D of C) In the AIX Graphics Library, a cubic spline approximation to a set of four control points that passes through the first and fourth control points and that has a continuous slope where two spline segments meet. Named after Dr. P. Bézier.

background - (1) In multiprogramming, the conditions under which low-priority programs are executed. Contrast with *foreground* . (2) An active session that is not currently displayed on the screen.

background color - The color in which the background of a graphic primitive is drawn.

background mix - An attribute that determines how the background of a graphic primitive is combined with the existing color of the graphics presentation space. Contrast with *mix* .

background program - In multiprogramming, a program that executes with a low priority. Contrast with *foreground program* .

bit map - A representation in memory of the data displayed on an APA device, usually the screen.

block - (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or logical records. (T) (2) To record data in a block. (3) A collection of contiguous records recorded as a unit. Blocks are separated by interblock gaps and each block may contain one or more records. (A)

block device - A storage device that performs I/O operations on blocks of data called *sectors* . Data on block devices can be randomly accessed. Block devices are designated by a drive letter (for example, **C:**).

blocking mode - A condition set by an application that determines when its threads might block. For example, an application might set the *Pipemode* parameter for the *DosCreateNPipe* function so that its threads perform I/O operations to the named pipe block when no data is available.

border - A visual indication (for example, a separator line or a background color) of the boundaries of a window.

boundary determination - An operation used to compute the size of the smallest rectangle that encloses a graphics object on the screen.

breakpoint - (1) A point in a computer program where execution may be halted. A breakpoint is usually at the beginning of an instruction where halts, caused by external intervention, are convenient for resuming execution. (T) (2) A place in a program, specified by a command or a condition, where the system halts execution and gives control to the workstation user or to a specified program.

broken pipe - When all of the handles that access one end of a pipe have been closed.

bucket - One or more fields in which the result of an operation is kept.

buffer - (1) A portion of storage used to hold input or output data temporarily. (2) To allocate and schedule the use of buffers. (A)

button - A mechanism used to request or initiate an action. See also *barrel buttons* , *bezel buttons* , *mouse button* , *push button* , and *radio button* .

byte pipe - Pipes that handle data as byte streams. All unnamed pipes are byte pipes. Named pipes can be byte pipes or message pipes. See *byte stream* .

byte stream - Data that consists of an unbroken stream of bytes.

Glossary - C

cache - A high-speed buffer storage that contains frequently accessed instructions and data; it is used to reduce access time.

cached micro presentation space - A presentation space from a Presentation-Manager-owned store of micro presentation spaces. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

CAD - Computer-Aided Design.

call - (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) To transfer control to a procedure, program, routine, or subroutine.

calling sequence - A sequence of instructions together with any associated data necessary to execute a call. (T)

Cancel - An action that removes the current window or menu without processing it, and returns the previous window.

cascaded menu - In the OS/2 operating system, a menu that appears when the arrow to the right of a cascading choice is selected. It contains a set of choices that are related to the cascading choice. Cascaded menus are used to reduce the length of a menu. See also *cascading choice* .

cascading choice - In SAA Common User Access architecture, a choice in a menu that, when selected, produces a cascaded menu containing other choices. An arrow () appears to the right of the cascading choice.

CASE statement - In PM programming, provides the body of a window procedure. There is usually one CASE statement for each message type supported by an application.

CGA - Color graphics adapter.

chained list - A list in which the data elements may be dispersed but in which each data element contains information for locating the next.
(T) Synonymous with *linked list* .

character - A letter, digit, or other symbol.

character box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. See also *character mode* . Contrast with *character cell* .

character cell - The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box* .

character code - The means of addressing a character in a character set, sometimes called *code point* .

character device - A device that performs I/O operations on one character at a time. Because character devices view data as a stream of bytes, character-device data cannot be randomly accessed. Character devices include the keyboard, mouse, and printer, and are referred to by name.

character mode - A mode that, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

character set - (1) An ordered set of unique representations called characters; for example, the 26 letters of English alphabet, Boolean 0 and 1, the set of symbols in the Morse code, and the 128 ASCII characters. (A) (2) All the valid characters for a programming language or for a computer system. (3) A group of characters used for a specific reason; for example, the set of characters a printer can print.

check box - In SAA Advanced Common User Access architecture, a square box with associated text that represents a choice. When a user selects a choice, an **X** appears in the check box to indicate that the choice is in effect. The user can clear the check box by selecting the choice again. Contrast with *radio button* .

check mark - (1) (D of C) In SAA Advanced Common User Access architecture, a symbol that shows that a choice is currently in effect. (2) The symbol that is used to indicate a selected item on a pull-down menu.

child process - In the OS/2 operating system, a process started by another process, which is called the parent process. Contrast with *parent process* .

child window - A window that appears within the border of its parent window (either a primary window or another child window). When the parent window is resized, moved, or destroyed, the child window also is resized, moved, or destroyed; however, the child window can be moved or resized independently from the parent window, within the boundaries of the parent window. Contrast with *parent window* .

choice - (1) An option that can be selected. The choice can be presented as text, as a symbol (number or letter), or as an icon (a pictorial symbol). (2) (D of C) In SAA Common User Access architecture, an item that a user can select.

chord - (1) To press more than one button on a pointing device while the pointer is within the limits that the user has specified for the operating environment. (2) (D of C) In graphics, a short line segment whose end points lie on a circle. Chords are a means for producing a circular image from straight lines. The higher the number of chords per circle, the smoother the circular image.

class - A way of categorizing objects based on their behavior and shape. A class is, in effect, a definition of a generic object. In SOM, a class is a special kind of object that can manufacture other objects that all have a common shape and exhibit similar behavior (more precisely, all of the objects manufactured by a class have the same memory layout and share a common set of methods). New classes can be defined in terms of existing classes through a technique known as *inheritance* .

class method - A class method of class **<X>** is a method provided by the metaclass of class **<X>**. Class methods are executed without requiring any instances of class **<X>** to exist, and are frequently used to create instances. In System Object Model, an action that can be performed on a class object.

class object - In System Object Model, the run-time implementation of a class.

class style - The set of properties that apply to every window in a window class.

client - (1) A functional unit that receives shared services from a server. (T) (2) A user, as in a client process that uses a named pipe or queue that is created and owned by a server process.

client area - The part of the window, inside the border, that is below the menu bar. It is the user's work space, where a user types information and selects choices from selection fields. In primary windows, it is where an application programmer presents the objects that a user works on.

client program - An application that creates and manipulates instances of classes.

client window - The window in which the application displays output and receives input. This window is located inside the frame window, under the window title bar and any menu bar, and within any scroll bars.

clip limits - The area of the paper that can be reached by a printer or plotter.

clipboard - In SAA Common User Access architecture, an area of computer memory, or storage, that temporarily holds data. Data in the

clipboard is available to other applications.

clipping - In computer graphics, removing those parts of a display image that lie outside a given boundary. (I) (A)

clipping area - The area in which the window can paint.

clipping path - A clipping boundary in world-coordinate space.

clock tick - The minimum unit of time that the system tracks. If the system timer currently counts at a rate of X Hz, the system tracks the time every 1/X of a second. Also known as *time tick*.

CLOCK\$ - Character-device name reserved for the system clock.

code page - An assignment of graphic characters and control-function meanings to all code points.

code point - (1) Synonym for *character code*. (2) (D of C) A 1-byte code representing one of 256 potential characters.

code segment - An executable section of programming code within a load module.

color dithering - See *dithering*.

color graphics adapter (CGA) - An adapter that simultaneously provides four colors and is supported by all IBM Personal Computer and Personal System/2 models.

command - The name and parameters associated with an action that a program can perform.

command area - An area composed of a command field prompt and a command entry field.

command entry field - An entry field in which users type commands.

command line - On a display screen, a display line, sometimes at the bottom of the screen, in which only commands can be entered.

command mode - A state of a system or device in which the user can enter commands.

command prompt - A field prompt showing the location of the command entry field in a panel.

Common Programming Interface (CPI) - Definitions of those application development languages and services that have, or are intended to have, implementations on and a high degree of commonality across the SAA environments. One of the three SAA architectural areas. See also *Common User Access architecture*.

Common User Access (CUA) architecture - Guidelines for the dialog between a human and a workstation or terminal. One of the three SAA architectural areas. See also *Common Programming Interface*.

compile - To translate a program written in a higher-level programming language into a machine language program.

composite window - A window composed of other windows (such as a frame window, frame-control windows, and a client window) that are kept together as a unit and that interact with each other.

computer-aided design (CAD) - The use of a computer to design or change a product, tool, or machine, such as using a computer for drafting or illustrating.

COM1, COM2, COM3 - Character-device names reserved for serial ports 1 through 3.

CON - Character-device name reserved for the console keyboard and screen.

conditional cascaded menu - A pull-down menu associated with a menu item that has a cascade mini-push button beside it in an object's pop-up menu. The conditional cascaded menu is displayed when the user selects the mini-push button.

container - In SAA Common User Access architecture, an object that holds other objects. A folder is an example of a container object. See also *folder* and *object*.

contextual help - In SAA Common User Access Architecture, help that gives specific information about the item the cursor is on. The help is contextual because it provides information about a specific item as it is currently being used. Contrast with *extended help*.

contiguous - Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

control - In SAA Advanced Common User Access architecture, a component of the user interface that allows a user to select choices or type information; for example, a check box, an entry field, a radio button.

control area - A storage area used by a computer program to hold control information. (I) (A)

Control Panel - In the Presentation Manager, a program used to set up user preferences that act globally across the system.

Control Program - (1) The basic functions of the operating system, including DOS emulation and the support for keyboard, mouse, and video input/output. (2) A computer program designed to schedule and to supervise the execution of programs of a computer system. (I) (A)

control window - A window that is used as part of a composite window to perform simple input and output tasks. Radio buttons and check boxes are examples.

control word - An instruction within a document that identifies its parts or indicates how to format the document.

coordinate space - A two-dimensional set of points used to generate output on a video display or printer.

Copy - A choice that places onto the clipboard, a copy of what the user has selected. See also *Cut* and *Paste* .

correlation - The action of determining which element or object within a picture is at a given position on the display. This follows a *pick* operation.

coverpage window - A window in which the application's help information is displayed.

CPI - Common Programming Interface.

critical extended attribute - An extended attribute that is necessary for the correct operation of the system or a particular application.

critical section - (1) In programming languages, a part of an asynchronous procedure that cannot be executed simultaneously with a certain part of another asynchronous procedure. (l)

Note: Part of the other asynchronous procedure also is a critical section. (2) A section of code that is not reentrant; that is, code that can be executed by only one thread at a time.

CUA architecture - Common User Access architecture.

current position - In computer graphics, the position, in user coordinates, that becomes the starting point for the next graphics routine, if that routine does not explicitly specify a starting point.

cursor - A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed. Types of cursors include text cursors, graphics cursors, and selection cursors. Contrast with *pointer* and *input focus* .

Cut - In SAA Common User Access architecture, a choice that removes a selected object, or a part of an object, to the clipboard, usually compressing the space it occupied in a window. See also *Copy* and *Paste* .

Glossary - D

daisy chain - A method of device interconnection for determining interrupt priority by connecting the interrupt sources serially.

data segment - A nonexecutable section of a program module; that is, a section of a program that contains data definitions.

data structure - The syntactic structure of symbolic expressions and their storage-allocation characteristics. (T)

data transfer - The movement of data from one object to another by way of the clipboard or by direct manipulation.

DBCS - Double-byte character set.

DDE - Dynamic data exchange.

deadlock - (1) Unresolved contention for the use of a resource. (2) An error condition in which processing cannot continue because each of two elements of the process is waiting for an action by, or a response from, the other. (3) An impasse that occurs when multiple processes are waiting for the availability of a resource that will not become available because it is being held by another process that is in a similar wait state.

debug - To detect, diagnose, and eliminate errors in programs. (T)

decipoint - In printing, one tenth of a point. There are 72 points in an inch.

default procedure - A function provided by the Presentation Manager Interface that may be used to process standard messages from dialogs or windows.

default value - A value assumed when no value has been specified. Synonymous with assumed value. For example, in the graphics programming interface, the default line-type is 'solid'.

definition list - A type of list that pairs a term and its description.

delta - An application-defined threshold, or number of container items, from either end of the list.

descendant - See *child process* .

descriptive text - Text used in addition to a field prompt to give more information about a field.

Deselect all - A choice that cancels the selection of all of the objects that have been selected in that window.

Desktop Manager - In the Presentation Manager, a window that displays a list of groups of programs, each of which can be started or stopped.

desktop window - The window, corresponding to the physical device, against which all other types of windows are established.

detached process - A background process that runs independent of the parent process.

detent - A point on a slider that represents an exact value to which a user can move the slider arm.

device context - A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context* , *information device context* , *memory device context* , *metafile device context* , *queued device context* , and *screen device context* .

device driver - A file that contains the code needed to attach and use a device such as a display, printer, or plotter.

device space - (1) Coordinate space in which graphics are assembled after all GPI transformations have been applied. Device space is defined in device-specific units. (2) (D of C) In computer graphics, a space defined by the complete set of addressable points of a display device. (A)

dialog - The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box - In SAA Advanced Common User Access architecture, a movable window, fixed in size, containing controls that a user uses to provide information required by an application so that it can continue to process a user request. See also *message box* , *primary window* , *secondary window* . Also known as a *pop-up window* .

Dialog Box Editor - A WYSIWYG editor that creates dialog boxes for communicating with the application user.

dialog item - A component (for example, a menu or a button) of a dialog box. Dialog items are also used when creating dialog templates.

dialog procedure - A dialog window that is controlled by a window procedure. It is responsible for responding to all messages sent to the dialog window.

dialog tag language - A markup language used by the DTL compiler to create dialog objects.

dialog template - The definition of a dialog box, which contains details of its position, appearance, and window ID, and the window ID of each of its child windows.

direct device context - A logical description of a data destination that is a device other than the screen (for example, a printer or plotter), and where the output is not to go through the spooler. Its purpose is to satisfy queries. See also *device context* .

direct manipulation - The user's ability to interact with an object by using the mouse, typically by dragging an object around on the Desktop and dropping it on other objects.

direct memory access (DMA) - A technique for moving data directly between main storage and peripheral equipment without requiring processing of the data by the processing unit.(T)

directory - A type of file containing the names and controlling information for other files or other directories.

display point - Synonym for *pel* .

dithering - (1) The process used in color displays whereby every other pel is set to one color, and the intermediate pels are set to another. Together they produce the effect of a third color at normal viewing distances. This process can only be used on solid areas of color; it does not work, for example, on narrow lines. (2) (D of C) In computer graphics, a technique of interleaving dark and light pixels so that the resulting image looks smoothly shaded when viewed from a distance.

DMA - Direct memory access.

DOS Protect Mode Interface (DPMI) - An interface between protect mode and real mode programs.

double-byte character set (DBCS) - A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

doubleword - A contiguous sequence of bits or characters that comprises two computer words and is capable of being addressed as a unit. (A)

DPMI - DOS Protect Mode Interface.

drag - In SAA Common User Access, to use a pointing device to move an object; for example, clicking on a window border, and dragging it to

make the window larger.

dragging - (1) In computer graphics, moving an object on the display screen as if it were attached to the pointer. (2) (D of C) In computer graphics, moving one or more segments on a display surface by translating. (I) (A)

drawing chain - See *segment chain* .

drop - To fix the position of an object that is being dragged, by releasing the select button of the pointing device.

drop - To fix the position of an object that is being dragged, by releasing the select button of the pointing device. See also *drag* .

DTL - Dialog tag language.

dual-boot function - A feature of the OS/2 operating system that allows the user to start DOS from within the operating system, or an OS/2 session from within DOS.

duplex - Pertaining to communication in which data can be sent and received at the same time. Synonymous with *full duplex* .

dynamic data exchange (DDE) - A message protocol used to communicate between applications that share data. The protocol uses shared memory as the means of exchanging data between applications.

dynamic data formatting - A formatting procedure that enables you to incorporate text, bit maps or metafiles in an IPF window at execution time.

dynamic link library - A collection of executable programming code and data that is bound to an application at load time or run time, rather than during linking. The programming code and data in a dynamic link library can be shared by several applications simultaneously.

dynamic linking - The process of resolving external references in a program module at load time or run time rather than during linking.

dynamic segments - Graphics segments drawn in exclusive-OR mix mode so that they can be moved from one screen position to another without affecting the rest of the displayed picture.

dynamic storage - (1) A device that stores data in a manner that permits the data to move or vary with time such that the specified data is not always available for recovery. (A) (2) A storage in which the cells require repetitive application of control signals in order to retain stored data. Such repetitive application of the control signals is called a refresh operation. A dynamic storage may use static addressing or sensing circuits. (A) (3) See also *static storage* .

dynamic time slicing - Varies the size of the time slice depending on system load and paging activity.

dynamic-link module - A module that is linked at load time or run time.

Glossary - E

EBCDIC - Extended binary-coded decimal interchange code. A coded character set consisting of 8-bit coded characters (9 bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment.

edge-triggered - Pertaining to an event semaphore that is posted then reset before a waiting thread gets a chance to run. The semaphore is considered to be posted for the rest of that thread's waiting period; the thread does not have to wait for the semaphore to be posted again.

EGA - Extended graphics adapter.

element - An entry in a graphics segment that comprises one or more graphics orders and that is addressed by the element pointer.

EMS - Expanded Memory Specification.

encapsulation - Hiding an object's implementation, that is, its private, internal data and methods. Private variables and methods are accessible only to the object that contains them.

entry field - In SAA Common User Access architecture, an area where a user types information. Its boundaries are usually indicated. See also *selection field* .

entry panel - A defined panel type containing one or more entry fields and protected information such as headings, prompts, and explanatory text.

entry-field control - The component of a user interface that provides the means by which the application receives data entered by the user in an entry field. When it has the input focus, the entry field displays a flashing pointer at the position where the next typed character will go.

environment segment - The list of environment variables and their values for a process.

environment strings - ASCII text strings that define the value of environment variables.

environment variables - Variables that describe the execution environment of a process. These variables are named by the operating system or by the application. Environment variables named by the operating system are PATH, DPATH, INCLUDE, INIT, LIB, PROMPT, and TEMP. The values of environment variables are defined by the user in the CONFIG.SYS file, or by using the SET command at the OS/2 command prompt.

error message - An indication that an error has been detected. (A)

event semaphore - A semaphore that enables a thread to signal a waiting thread or threads that an event has occurred or that a task has been completed. The waiting threads can then perform an action that is dependent on the completion of the signaled event.

exception - An abnormal condition such as an I/O error encountered in processing a data set or a file.

exclusive system semaphore - A system semaphore that can be modified only by threads within the same process.

executable file - (1) A file that contains programs or commands that perform operations or actions to be taken. (2) A collection of related data records that execute programs.

exit - To execute an instruction within a portion of a computer program in order to terminate the execution of that portion. Such portions of computer programs include loops, subroutines, modules, and so on. (T) Repeated exit requests return the user to the point from which all functions provided to the system are accessible. Contrast with *cancel* .

expanded memory specification (EMS) - Enables DOS applications to access memory above the 1MB real mode addressing limit.

extended attribute - An additional piece of information about a file object, such as its data format or category. It consists of a name and a value. A file object may have more than one extended attribute associated with it.

extended help - In SAA Common User Access architecture, a help action that provides information about the contents of the application window from which a user requested help. Contrast with *contextual help* .

extended-choice selection - A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection* .

extent - Continuous space on a disk or diskette that is occupied by or reserved for a particular data set, data space, or file.

external link - In Information Presentation Facility, a link that connects external online document files.

Glossary - F

family-mode application - An application program that can run in the OS/2 environment and in the DOS environment; however, it cannot take advantage of many of the OS/2-mode facilities, such as multitasking, interprocess communication, and dynamic linking.

FAT - File allocation table.

FEA - Full extended attribute.

field-level help - Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used; the information is dependent upon the context within the work session.

FIFO - First-in-first-out. (A)

file - A named set of records stored or processed as a unit. (T)

file allocation table (FAT) - In IBM personal computers, a table used by the operating system to allocate space on a disk for a file, and to locate and chain together parts of the file that may be scattered on different sectors so that the file can be used in a random or sequential manner.

file attribute - Any of the attributes that describe the characteristics of a file.

File Manager - In the Presentation Manager, a program that displays directories and files, and allows various actions on them.

file specification - The full identifier for a file, which includes its drive designation, path, file name, and extension.

file system - The combination of software and hardware that supports storing information on a storage device.

file system driver (FSD) - A program that manages file I/O and controls the format of information on the storage media.

fillet - A curve that is tangential to the end points of two adjoining lines. See also *polyfillet* .

filtering - An application process that changes the order of data in a queue.

first-in-first-out (FIFO) - A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

flag - (1) An indicator or parameter that shows the setting of a switch. (2) A character that signals the occurrence of some condition, such as the end of a word. (A) (3) (D of C) A characteristic of a file or directory that enables it to be used in certain ways. See also *archive flag* , *hidden flag* , and *read-only flag* .

focus - See *input focus* .

folder - A container used to organize objects.

font - A particular size and style of typeface that contains definitions of character sets, marker sets, and pattern sets.

Font Editor - A utility program provided with the IBM Developers Toolkit that enables the design and creation of new fonts.

foreground program - (1) The program with which the user is currently interacting. Also known as *interactive program* . Contrast with *background program* . (2) (D of C) In multiprogramming, a high-priority program.

frame - The part of a window that can contain several different visual elements specified by the application, but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles - Standard window layouts provided by the Presentation Manager.

FSD - File system driver.

full-duplex - Synonym for *duplex* .

full-screen application - An application that has complete control of the screen.

function - (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a call. (2) A set of related control statements that cause one or more programs to be performed.

function key - A key that causes a specified sequence of operations to be performed when it is pressed, for example, F1 and Alt-K.

function key area - The area at the bottom of a window that contains function key assignments such as F1=Help.

Glossary - G

GDT - Global Descriptor Table.

general protection fault - An exception condition that occurs when a process attempts to use storage or a module that has some level of protection assigned to it, such as I/O privilege level. See also *IOPL code segment* .

Global Descriptor Table (GDT) - A table that defines code and data segments available to all tasks in an application.

global dynamic-link module - A dynamic-link module that can be shared by all processes in the system that refer to the module name.

global file-name character - Either a question mark (?) or an asterisk (*) used as a variable in a file name or file name extension when referring to a particular file or group of files.

glyph - A graphic symbol whose appearance conveys information.

GPI - Graphics programming interface.

graphic primitive - In computer graphics, a basic element, such as an arc or a line, that is not made up of smaller parts and that is used to create diagrams and pictures. See also *graphics segment* .

graphics - (1) A picture defined in terms of graphic primitives and graphics attributes. (2) (D of C) The making of charts and pictures. (3) Pertaining to charts, tables, and their creation. (4) See *computer graphics*, *coordinate graphics*, *fixed-image graphics*, *interactive graphics*, *passive graphics*, *raster graphics* .

graphics attributes - Attributes that apply to graphic primitives. Examples are color, line type, and shading-pattern definition. See also *segment attributes* .

graphics field - The clipping boundary that defines the visible part of the presentation-page contents.

graphics mode - One of several states of a display. The mode determines the resolution and color content of the screen.

graphics model space - The conceptual coordinate space in which a picture is constructed after any model transforms have been applied.
Also known as *model space* .

Graphics programming interface - The formally defined programming language that is between an IBM graphics program and the user of the program.

graphics segment - A sequence of related graphic primitives and graphics attributes. See also *graphic primitive* .

graying - The indication that a choice on a pull-down is unavailable.

group - A collection of logically connected controls. For example, the buttons controlling paper size for a printer could be called a group. See also *program group* .

Glossary - H

handle - (1) An identifier that represents an object, such as a device or window, to the Presentation Interface. (2) (D of C) In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

hard error - An error condition on a network that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

header - (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message, such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

heading tags - A document element that enables information to be displayed in windows, and that controls entries in the contents window controls placement of push buttons in a window, and defines the shape and size of windows.

heap - An area of free storage available for dynamic allocation by an application. Its size varies according to the storage requirements of the application.

help function - (1) A function that provides information about a specific field, an application panel, or information about the help facility. (2) (D of C) One or more display images that describe how to use application software or how to do a system operation.

Help index - In SAA Common User Access architecture, a help action that provides an index of the help information available for an application.

help panel - A panel with information to assist users that is displayed in response to a help request from the user.

help window - A Common-User-Access-defined secondary window that displays information when the user requests help.

hidden file - An operating system file that is not displayed by a directory listing.

hide button - In the OS/2 operating system, a small, square button located in the right-hand corner of the title bar of a window that, when selected, removes from the screen all the windows associated with that window. Contrast with *maximize button* . See also *restore button* .

hierarchical inheritance - The relationship between parent and child classes. An object that is lower in the inheritance hierarchy than another object, inherits all the characteristics and behaviors of the objects above it in the hierarchy.

hierarchy - A tree of segments beginning with the root segment and proceeding downward to dependent segment types.

high-performance file system (HPFS) - In the OS/2 operating system, an installable file system that uses high-speed buffer storage, known as a cache, to provide fast access to large disk volumes. The file system also supports the coexistence of multiple, active file systems on a single personal computer, with the capability of multiple and different storage devices. File names used with the HPFS can have as many as 254 characters.

hit testing - The means of identifying which window is associated with which input device event.

hook - A point in a system-defined function where an application can supply additional code that the system processes as though it were part of the function.

hook chain - A sequence of hook procedures that are "chained" together so that each event is passed, in turn, to each procedure in the chain.

hot spot - The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer. Contrast with *action point* .

HPFS - high-performance file system.

hypergraphic link - A connection between one piece of information and another through the use of graphics.

hypertext - A way of presenting information online with connections between one piece of information and another, called *hypertext links* .
See also *hypertext link* .

hypertext link - A connection between one piece of information and another.

Glossary - I

I/O operation - An input operation to, or output operation from a device attached to a computer.

I-beam pointer - A pointer that indicates an area, such as an entry field in which text can be edited.

icon - In SAA Advanced Common User Access architecture, a graphical representation of an object, consisting of an image, image background, and a label. Icons can represent items (such as a document file) that the user wants to work on, and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions, and minimized programs.

icon area - In the Presentation Manager, the area at the bottom of the screen that is normally used to display the icons for minimized programs.

Icon Editor - The Presentation Manager-provided tool for creating icons.

IDL - Interface Definition Language.

image font - A set of symbols, each of which is described in a rectangular array of pels. Some of the pels in the array are set to produce the image of one of the symbols. Contrast with *outline font* .

implied metaclass - Subclassing the metaclass of a parent class without a separate CSC for the resultant metaclass.

indirect manipulation - Interaction with an object through choices and controls.

information device context - A logical description of a data destination other than the screen (for example, a printer or plotter), but where no output will occur. Its purpose is to satisfy queries. See also *device context* .

information panel - A defined panel type characterized by a body containing only protected information.

Information Presentation Facility (IPF) - A facility provided by the OS/2 operating system, by which application developers can produce online documentation and context-sensitive online help panels for their applications.

inheritance - The technique of specifying the shape and behavior of one class (called a *subclass*) as incremental differences from another class (called the *parent class* or *superclass*). The subclass inherits the superclass' state representation and methods, and can provide additional data elements and methods. The subclass also can provide new functions with the same method names used by the superclass. Such a subclass method is said to override the superclass method, and will be selected automatically by method resolution on subclass instances. An overriding method can elect to call upon the superclass' method as part of its own implementation.

input focus - (1) The area of a window where user interaction is possible using an input device, such as a mouse or the keyboard. (2) The position in the *active window* where a user's normal interaction with the keyboard will appear.

input router - An internal OS/2 process that removes messages from the system queue.

input/output control - A device-specific command that requests a function of a device driver.

installable file system (IFS) - A file system in which software is installed when the operating system is started.

instance - (Or object instance). A specific object, as distinguished from the abstract definition of an object referred to as its class.

instance method - A method valid for a particular object.

instruction pointer - In System/38, a pointer that provides addressability for a machine interface instruction in a program.

integer atom - An *atom* that represents a predefined system constant and carries no storage overhead. For example, names of window classes provided by Presentation Manager are expressed as integer atoms.

interactive graphics - Graphics that can be moved or manipulated by a user at a terminal.

interactive program - (1) A program that is running (active) and is ready to receive (or is receiving) input from a user. (2) A running program that can receive input from the keyboard or another input device. Compare with *active program* and contrast with *noninteractive program*.

Also known as a *foreground program*.

interchange file - A file containing data that can be sent from one Presentation Manager interface application to another.

Interface Definition Language (IDL) - Language-neutral interface specification for a SOM class.

interpreter - A program that translates and executes each instruction of a high-level programming language before it translates and executes.

interprocess communication (IPC) - In the OS/2 operating system, the exchange of information between processes or threads through semaphores, pipes, queues, and shared memory.

interval timer - (1) A timer that provides program interruptions on a program-controlled basis. (2) An electronic counter that counts intervals of time under program control.

IOCtl - Input/output control.

IOPL - Input/output privilege level.

IOPL code segment - An IOPL executable section of programming code that enables an application to directly manipulate hardware interrupts and ports without replacing the device driver. See also *privilege level*.

IPC - Interprocess communication.

IPF - Information Presentation Facility.

IPF compiler - A text compiler that interpret tags in a source file and converts the information into the specified format.

IPF tag language - A markup language that provides the instructions for displaying online information.

item - A data object that can be passed in a DDE transaction.

Glossary - J

journal - A special-purpose file that is used to record changes made in the system.

Glossary - K

Kanji - A graphic character set used in Japanese ideographic alphabets.

KBD\$ - Character-device name reserved for the keyboard.

kernel - The part of an operating system that performs basic functions, such as allocating hardware resources.

Kerning - The design of graphics characters so that their character boxes overlap. Used to space text proportionally.

keyboard accelerator - A keystroke that generates a command message for an application.

keyboard augmentation - A function that enables a user to press a keyboard key while pressing a mouse button.

keyboard focus - A temporary attribute of a window. The window that has a keyboard focus receives all keyboard input until the focus changes to a different window.

Keys help - In SAA Common User Access architecture, a help action that provides a listing of the application keys and their assigned functions.

Glossary - L

label - In a graphics segment, an identifier of one or more elements that is used when editing the segment.

LAN - local area network.

language support procedure - A function provided by the Presentation Manager Interface for applications that do not, or cannot (as in the case of COBOL and FORTRAN programs), provide their own dialog or window procedures.

lazy drag - See *pickup and drop* .

lazy drag set - See *pickup set* .

LDT - In the OS/2 operating system, Local Descriptor Table.

LIFO stack - A stack from which data is retrieved in last-in, first-out order.

linear address - A unique value that identifies the memory object.

linked list - Synonym for *chained list* .

list box - In SAA Advanced Common User Access architecture, a control that contains scrollable choices from which a user can select one choice.

Note: In CUA architecture, this is a programmer term. The end user term is selection list.

list button - A button labeled with an underlined down-arrow that presents a list of valid objects or choices that can be selected for that field.

list panel - A defined panel type that displays a list of items from which users can select one or more choices and then specify one or more actions to work on those choices.

load time - The point in time at which a program module is loaded into main storage for execution.

load-on-call - A function of a linkage editor that allows selected segments of the module to be disk resident while other segments are executing. Disk resident segments are loaded for execution and given control when any entry point that they contain is called.

local area network (LAN) - (1) A computer network located on a user's premises within a limited geographical area. Communication within a local area network is not subject to external regulations; however, communication across the LAN boundary may be subject to some form of regulation. (T)

Note: A LAN does not use store and forward techniques. (2) A network in which a set of devices are connected to one another for communication and that can be connected to a larger network.

Local Descriptor Table (LDT) - Defines code and data segments specific to a single task.

lock - A serialization mechanism by means of which a resource is restricted for use by the holder of the lock.

logical storage device - A device that the user can map to a physical (actual) device.

LPT1, LPT2, LPT3 - Character-device names reserved for parallel printers 1 through 3.

Glossary - M

main window - The window that is positioned relative to the *desktop window* .

manipulation button - The button on a pointing device a user presses to directly manipulate an object.

map - (1) A set of values having a defined correspondence with the quantities or values of another set. (I) (A) (2) To establish a set of values having a defined correspondence with the quantities or values of another set. (I)

marker box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single marker from a marker set.

marker symbol - A symbol centered on a point. Graphs and charts can use marker symbols to indicate the plotted points.

marquee box - The rectangle that appears during a selection technique in which a user selects objects by drawing a box around them with a pointing device.

Master Help Index - In the OS/2 operating system, an alphabetic list of help topics related to using the operating system.

maximize - To enlarge a window to its largest possible size.

media window - The part of the physical device (display, printer, or plotter) on which a picture is presented.

memory block - Part memory within a heap.

memory device context - A logical description of a data destination that is a memory bit map. See also *device context* .

memory management - A feature of the operating system for allocating, sharing, and freeing main storage.

memory object - Logical unit of memory requested by an application, which forms the granular unit of memory manipulation from the application viewpoint.

menu - In SAA Advanced Common User Access architecture, an extension of the menu bar that displays a list of choices available for a selected choice in the menu bar. After a user selects a choice in menu bar, the corresponding menu appears. Additional pop-up windows can appear from menu choices.

menu bar - In SAA Advanced Common User Access architecture, the area near the top of a window, below the title bar and above the rest of the window, that contains choices that provide access to other menus.

menu button - The button on a pointing device that a user presses to view a pop-up menu associated with an object.

message - (1) In the Presentation Manager, a packet of data used for communication between the Presentation Manager interface and Presentation Manager applications (2) In a user interface, information not requested by users but presented to users by the computer in response to a user action or internal process.

message box - (1) A dialog window predefined by the system and used as a simple interface for applications, without the necessity of creating dialog-template resources or dialog procedures. (2) (D of C) In SAA Advanced Common User Access architecture, a type of window that shows messages to users. See also *dialog box*, *primary window*, *secondary window* .

message filter - The means of selecting which messages from a specific window will be handled by the application.

message queue - A sequenced collection of messages to be read by the application.

message stream mode - A method of operation in which data is treated as a stream of messages. Contrast with *byte stream* .

metacharacter - See *global file-name character* .

metaclass - A class whose instances are all classes. In SOM, any class descended from SOMClass is a metaclass. The methods of a metaclass are sometimes called "class" methods.

metafile - A file containing a series of attributes that set color, shape and size, usually of a picture or a drawing. Using a program that can interpret these attributes, a user can view the assembled image.

metafile device context - A logical description of a data destination that is a metafile, which is used for graphics interchange. See also *device context* .

metalanguage - A language used to specify another language. For example, data types can be described using a metalanguage so as to make the descriptions independent of any one computer language.

method - One of the units that makes up the behavior of an object. A method is a combination of a function and a name, such that many different functions can have the same name. Which function the name refers to at any point in time depends on the object that is to execute the method and is the subject of method resolution.

method override - The replacement, by a child class, of the implementation of a method inherited from a parent and an ancestor class.

mickey - A unit of measurement for physical mouse motion whose value depends on the mouse device driver currently loaded.

micro presentation space - A graphics presentation space in which a restricted set of the GPI function calls is available.

minimize - To remove from the screen all windows associated with an application and replace them with an icon that represents the application.

mix - An attribute that determines how the foreground of a graphic primitive is combined with the existing color of graphics output. Also known as *foreground mix* . Contrast with *background mix* .

mixed character string - A string containing a mixture of one-byte and *Kanji* or Hangeul (two-byte) characters.

mnemonic - (1) A method of selecting an item on a pull-down by means of typing the highlighted letter in the menu item. (2) (D of C) In SAA Advanced Common User Access architecture, usually a single character, within the text of a choice, identified by an underscore beneath the character. If all characters in a choice already serve as mnemonics for other choices, another character, placed in parentheses immediately following the choice, can be used. When a user types the mnemonic for a choice, the choice is either selected or the cursor is moved to that choice.

modal dialog box - In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that requires a user to enter information before continuing to work in the application window from which it was displayed. Contrast with *modeless dialog box* . Also known as a *serial dialog box* . Contrast with *parallel dialog box* .

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

model space - See *graphics model space* .

modeless dialog box - In SAA Advanced Common User Access architecture, a type of movable window, fixed in size, that allows users to continue their dialog with the application without entering information in the dialog box. Also known as a *parallel dialog box* . Contrast with *modal dialog box* .

Note: In CUA architecture, this is a programmer term. The end user term is pop-up window.

module definition file - A file that describes the code segments within a load module. For example, it indicates whether a code segment is loadable before module execution begins (preload), or loadable only when referred to at run time (load-on-call).

mouse - In SAA usage, a device that a user moves on a flat surface to position a pointer on the screen. It allows a user to select a choice or function to be performed or to perform operations on the screen, such as dragging or drawing lines from one position to another.

MOUSE\$ - Character-device name reserved for a mouse.

multiple-choice selection - In SAA Basic Common User Access architecture, a type of field from which a user can select one or more choices or select none. See also *check box* . Contrast with *extended-choice selection* .

multiple-line entry field - In SAA Advanced Common User Access architecture, a control into which a user types more than one line of information. See also *single-line entry field* .

multitasking - The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

mutex semaphore - (Mutual exclusion semaphore). A semaphore that enables threads to serialize their access to resources. Only the thread that currently owns the mutex semaphore can gain access to the resource, thus preventing one thread from interrupting operations being performed by another.

muxwait semaphore - (Multiple wait semaphore). A semaphore that enables a thread to wait either for multiple event semaphores to be posted or for multiple mutex semaphores to be released. Alternatively, a muxwait semaphore can be set to enable a thread to wait for any ONE of the event or mutex semaphores in the muxwait semaphore's list to be posted or released.

Glossary - N

named pipe - A named buffer that provides client-to-server, server-to-client, or full duplex communication between unrelated processes. Contrast with *unnamed pipe* .

national language support (NLS) - The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, MRI, or documentation of a product.

nested list - A list that is contained within another list.

NLS - national language support.

non-8.3 file-name format - A file-naming convention in which file names can consist of up to 255 characters. See also *8.3 file-name format* .

noncritical extended attribute - An extended attribute that is not necessary for the function of an application.

nondestructive read - Reading that does not erase the data in the source location. (T)

noninteractive program - A running program that cannot receive input from the keyboard or other input device. Compare with *active program*, and contrast with *interactive program* .

nonretained graphics - Graphic primitives that are not remembered by the Presentation Manager interface when they have been drawn.

Contrast with *retained graphics* .

null character (NUL) - (1) Character-device name reserved for a nonexistent (dummy) device. (2) (D of C) A control character that is used to accomplish media-fill or time-fill and that may be inserted into or removed from a sequence of characters without affecting the meaning of the sequence; however, the control of equipment or the format may be affected by this character. (I) (A)

null-terminated string - A string of (n+1) characters where the (n+1)th character is the 'null' character (0x00) Also known as 'zero-terminated' string and 'ASCIIZ' string.

Glossary - O

object - The elements of data and function that programs create, manipulate, pass as arguments, and so forth. An object is a way of associating specific data values with a specific set of named functions (called *methods*) for a period of time (referred to as the *lifetime* of the object). The data values of an object are referred to as its *state* . In SOM, objects are created by other objects called *classes* . The specification of what comprises the set of functions and data elements that make up an object is referred to as the *definition* of a class.

SOM objects offer a high degree of *encapsulation* . This property permits many aspects of the implementation of an object to change without affecting client programs that depend on the object's behavior.

object definition - See *class* .

object instance - See *instance* .

Object Interface Definition Language (OIDL) - Specification language used in SOM Version 1 for defining classes. Replaced by Interface Definition Language (IDL).

object window - A window that does not have a parent but which might have child windows. An object window cannot be presented on a device.

OIDL - Object Interface Definition Language.

open - To start working with a file, directory, or other object.

ordered list - Vertical arrangements of items, with each item in the list preceded by a number or letter.

outline font - A set of symbols, each of which is created as a series of lines and curves. Synonymous with *vector font* . Contrast with *image font* .

output area - An area of storage reserved for output. (A)

owner window - A window into which specific events that occur in another (owned) window are reported.

ownership - The determination of how windows communicate using messages.

owning process - The process that owns the resources that might be shared with other processes.

Glossary - P

page - (1) A 4KB segment of contiguous physical memory. (2) (D of C) A defined unit of space on a storage medium.

page viewport - A boundary in device coordinates that defines the area of the output device in which graphics are to be displayed. The presentation-page contents are transformed automatically to the page viewport in device space.

paint - (1) The action of drawing or redrawing the contents of a window. (2) In computer graphics, to shade an area of a display image; for example, with crosshatching or color.

panel - In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

panel area - An area within a panel that contains related information. The three major Common User Access-defined panel areas are the action bar, the function key area, and the panel body.

panel area separator - In SAA Basic Common User Access architecture, a solid, dashed, or blank line that provides a visual distinction between two adjacent areas of a panel.

panel body - The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body can contain protected information, selection fields, and entry fields. The layout and content of the panel body determine the panel type.

panel body area - See *client area* .

panel definition - A description of the contents and characteristics of a panel. A panel definition is the application developer's mechanism for predefining the format to be presented to users in a window.

panel ID - In SAA Basic Common User Access architecture, a panel identifier, located in the upper-left corner of a panel. A user can choose whether to display the panel ID.

panel title - In SAA Basic Common User Access architecture, a particular arrangement of information that is presented in a window or pop-up. If some of the information is not visible, a user can scroll through the information.

paper size - The size of paper, defined in either standard U.S. or European names (for example, A, B, A4), and measured in inches or millimeters respectively.

parallel dialog box - See *modeless dialog box* .

parameter list - A list of values that provides a means of associating addressability of data defined in a called program with data in the calling program. It contains parameter names and the order in which they are to be associated in the calling and called program.

parent class - See *inheritance* .

parent process - In the OS/2 operating system, a process that creates other processes. Contrast with *child process* .

parent window - In the OS/2 operating system, a window that creates a child window. The child window is drawn within the parent window. If the parent window is moved, resized, or destroyed, the child window also will be moved, resized, or destroyed. However, the child window can be moved and resized independently from the parent window, within the boundaries of the parent window. Contrast with *child window* .

partition - (1) A fixed-size division of storage. (2) On an IBM personal computer fixed disk, one of four possible storage areas of variable size; one may be accessed by DOS, and each of the others may be assigned to another operating system.

Paste - A choice in the Edit pull-down that a user selects to move the contents of the clipboard into a preselected location. See also *Copy* and *Cut* .

path - The route used to locate files; the storage location of a file. A fully qualified path lists the drive identifier, directory name, subdirectory name (if any), and file name with the associated extension.

PDD - Physical device driver.

peeking - An action taken by any thread in the process that owns the queue to examine queue elements without removing them.

pel - (1) The smallest area of a display screen capable of being addressed and switched between visible and invisible states. Synonym for *display point* , *pixel* , and *picture element* . (2) (D of C) Picture element.

persistent object - An object whose instance data and state are preserved between system shutdown and system startup.

physical device driver (PDD) - A system interface that handles hardware interrupts and supports a set of input and output functions.

pick - To select part of a displayed object using the pointer.

pickup - To add an object or set of objects to the pickup set.

pickup and drop - A drag operation that does not require the direct manipulation button to be pressed for the duration of the drag.

pickup set - The set of objects that have been picked up as part of a pickup and drop operation.

picture chain - See *segment chain* .

picture element - (1) Synonym for *pel* . (2) (D of C) In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity. (T) . (3) The area of the finest detail that can be reproduced effectively on the recording medium.

PID - Process identification.

pipe - (1) A named or unnamed buffer used to pass data between processes. A process reads from or writes to a pipe as if the pipe were a standard-input or standard-output file. See also *named pipe* and *unnamed pipe* . (2) (D of C) To direct data so that the output from one process becomes the input to another process. The standard output of one command can be connected to the standard input of another with the pipe operator (*|*).

pixel - (1) Synonym for *pel* . (2) (D of C) Picture element.

plotter - An output unit that directly produces a hardcopy record of data on a removable medium, in the form of a two-dimensional graphic representation. (T)

PM - Presentation Manager.

pointer - (1) The symbol displayed on the screen that is moved by a pointing device, such as a *mouse* . The pointer is used to point at items that users can select. Contrast with *cursor* . (2) A data element that indicates the location of another data element. (T)

POINTERS\$ - Character-device name reserved for a pointer device (mouse screen support).

pointing device - In SAA Advanced Common User Access architecture, an instrument, such as a mouse, trackball, or joystick, used to move a pointer on the screen.

pointings - Pairs of x-y coordinates produced by an operator defining positions on a screen with a pointing device, such as a *mouse* .

polyfillet - A curve based on a sequence of lines. The curve is tangential to the end points of the first and last lines, and tangential also to the midpoints of all other lines. See also *fillet* .

polygon - One or more closed figures that can be drawn filled, outlined, or filled and outlined.

polyline - A sequence of adjoining lines.

polymorphism - The ability to have different implementations of the same method for two or more classes of objects.

pop - To retrieve an item from a last-in-first-out stack of items. Contrast with *push* .

pop-up menu - A menu that lists the actions that a user can perform on an object. The contents of the pop-up menu can vary depending on the context, or state, of the object.

pop-up window - (1) A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window. (2) (D of C) In SAA Advanced Common User Access architecture, a movable window, fixed in size, in which a user provides information required by an application so that it can continue to process a user request.

presentation drivers - Special purpose I/O routines that handle field device-independent I/O requests from the PM and its applications.

Presentation Manager (PM) - The interface of the OS/2 operating system that presents, in windows a graphics-based interface to applications and files installed and running under the OS/2 operating system.

presentation page - The coordinate space in which a picture is assembled for display.

presentation space (PS) - (1) Contains the device-independent definition of a picture. (2) (D of C) The display space on a display device.

primary window - In SAA Common User Access architecture, the window in which the main interaction between the user and the application takes place. In a multiprogramming environment, each application starts in its own primary window. The primary window remains for the duration of the application, although the panel displayed will change as the user's dialog moves forward. See also *secondary window* .

primitive - In computer graphics, one of several simple functions for drawing on the screen, including, for example, the rectangle, line, ellipse, polygon, and so on.

primitive attribute - A specifiable characteristic of a graphic primitive. See *graphics attributes* .

print job - The result of sending a document or picture to be printed.

Print Manager - In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows users to view print queues and to manipulate print jobs.

privilege level - A protection level imposed by the hardware architecture of the IBM personal computer. There are four privilege levels (number 0 through 3). Only certain types of programs are allowed to execute at each privilege level. See also *IOPL code segment* .

procedure call - In programming languages, a language construct for invoking execution of a procedure.

process - An instance of an executing application and the resources it is using.

program - A sequence of instructions that a computer can interpret and execute.

program details - Information about a program that is specified in the *Program Manager* window and is used when the program is started.

program group - In the Presentation Manager, several programs that can be acted upon as a single entity.

program name - The full file specification of a program. Contrast with *program title* .

program title - The name of a program as it is listed in the *Program Manager* window. Contrast with *program name* .

prompt - A displayed symbol or message that requests input from the user or gives operational information; for example, on the display screen of an IBM personal computer, the DOS A> prompt. The user must respond to the prompt in order to proceed.

protect mode - A method of program operation that limits or prevents access to certain instructions or areas of storage. Contrast with *real mode*.

protocol - A set of semantic and syntactic rules that determines the behavior of functional units in achieving communication. (I)

pseudocode - An artificial language used to describe computer program algorithms without using the syntax of any particular programming language. (A)

pull-down - (1) An *action bar* extension that displays a list of choices available for a selected action bar choice. After users select an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the actions available to users. (2) (D of C) In SAA Common User Access architecture, pertaining to a choice in an action bar pull-down.

push - To add an item to a last-in-first-out stack of items. Contrast with *pop*.

push button - In SAA Advanced Common User Access architecture, a rectangle with text inside. Push buttons are used in windows for actions that occur immediately when the push button is selected.

putback - To remove an object or set of objects from the lazy drag set. This has the effect of undoing the pickup operation for those objects

putdown - To drop the objects in the lazy drag set on the target object.

Glossary - Q

queue - (1) A linked list of elements waiting to be processed in FIFO order. For example, a queue may be a list of print jobs waiting to be printed. (2) (D of C) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed.

queued device context - A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

Glossary - R

radio button - (1) A control window, shaped like a round button on the screen, that can be in a checked or unchecked state. It is used to select a single item from a list. Contrast with *check box*. (2) In SAA Advanced Common User Access architecture, a circle with text beside it. Radio buttons are combined to show a user a fixed set of choices from which only one can be selected. The circle is partially filled when a choice is selected.

RAS - Reliability, availability, and serviceability.

raster - (1) In computer graphics, a predetermined pattern of lines that provides uniform coverage of a display space. (T) (2) The coordinate grid that divides the display area of a display device. (A)

read-only file - A file that can be read from but not written to.

real mode - A method of program operation that does not limit or prevent access to any instructions or areas of storage. The operating system loads the entire program into storage and gives the program access to all system resources. Contrast with *protect mode*.

realize - To cause the system to ensure, wherever possible, that the physical color table of a device is set to the closest possible match in the logical color table.

recursive routine - A routine that can call itself, or be called by another routine that was called by the recursive routine.

reentrant - The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase - (1) A word or phrase that is emphasized in a device-dependent manner to inform the user that additional information for the word or phrase is available. (2) (D of C) In hypertext, text that is highlighted and preceded by a single-character input field used to signify the existence of a hypertext link.

reference phrase help - In SAA Common User Access architecture, highlighted words or phrases within help information that a user selects to get additional information.

refresh - To update a window, with changed information, to its current status.

region - A clipping boundary in device space.

register - A part of internal storage having a specified storage capacity and usually intended for a specific purpose. (T)

remote file system - A file-system driver that gains access to a remote system without a block device driver.

resource - The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators, and mnemonics; the definitions are held in a resource file.

resource file - A file containing information used in the definition of a window. Definitions can be of fonts, templates, accelerators, and mnemonics.

restore - To return a window to its original size or position following a sizing or moving action.

retained graphics - Graphic primitives that are remembered by the Presentation Manager interface after they have been drawn. Contrast with *nonretained graphics* .

return code - (1) A value returned to a program to indicate the results of an operation requested by that program. (2) A code used to influence the execution of succeeding instructions.(A)

reverse video - (1) A form of highlighting a character, field, or cursor by reversing the color of the character, field, or cursor with its background; for example, changing a red character on a black background to a black character on a red background. (2) In SAA Basic Common User Access architecture, a screen emphasis feature that interchanges the foreground and background colors of an item.

REXX Language - Restructured Extended Executor. A procedural language that provides batch language functions along with structured programming constructs such as loops; conditional testing and subroutines.

RGB - (1) Color coding in which the brightness of the additive primary colors of light, red, green, and blue, are specified as three distinct values of white light. (2) Pertaining to a color display that accepts signals representing red, green, and blue.

roman - Relating to a type style with upright characters.

root segment - In a hierarchical database, the highest segment in the tree structure.

round-robin scheduling - A process that allows each thread to run for a specified amount of time.

run time - (1) Any instant at which the execution of a particular computer program takes place. (T) (2) The amount of time needed for the execution of a particular computer program. (T) (3) The time during which an instruction in an instruction register is decoded and performed. Synonym for *execution time* .

Glossary - S

SAA - Systems Application Architecture.

SBCS - Single-byte character set.

scheduler - A computer program designed to perform functions such as scheduling, initiation, and termination of jobs.

screen - In SAA Basic Common User Access architecture, the physical surface of a display device upon which information is shown to a user.

screen device context - A logical description of a data destination that is a particular window on the screen. See also *device context* .

SCREEN\$ - Character-device name reserved for the display screen.

scroll bar - In SAA Advanced Common User Access architecture, a part of a window, associated with a scrollable area, that a user interacts with to see information that is not currently allows visible.

scrollable entry field - An entry field larger than the visible field.

scrollable selection field - A selection field that contains more choices than are visible.

scrolling - Moving a display image vertically or horizontally in a manner such that new data appears at one edge, as existing data disappears at the opposite edge.

secondary window - A window that contains information that is dependent on information in a primary window and is used to supplement the interaction in the primary window.

sector - On disk or diskette storage, an addressable subdivision of a track used to record one block of a program or data.

segment - See *graphics segment* .

segment attributes - Attributes that apply to the segment as an entity, as opposed to the individual primitives within the segment. For example, the visibility or detectability of a segment.

segment chain - All segments in a graphics presentation space that are defined with the 'chained' attribute. Synonym for *picture chain* .

segment priority - The order in which segments are drawn.

segment store - An area in a normal graphics presentation space where retained graphics segments are stored.

select - To mark or choose an item. Note that *select* means to mark or type in a choice on the screen; *enter* means to send all selected choices to the computer for processing.

select button - The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also known as button 1.

selection cursor - In SAA Advanced Common User Access architecture, a visual indication that a user has selected a choice. It is represented by outlining the choice with a dotted box. See also *text cursor* .

selection field - (1) In SAA Advanced Common User Access architecture, a set of related choices. See also *entry field* . (2) In SAA Basic Common User Access architecture, an area of a panel that cannot be scrolled and contains a fixed number of choices.

semantics - The relationships between symbols and their meanings.

semaphore - An object used by applications for signalling purposes and for controlling access to serially reusable resources.

separator - In SAA Advanced Common User Access architecture, a line or color boundary that provides a visual distinction between two adjacent areas.

serial dialog box - See *modal dialog box* .

serialization - The consecutive ordering of items.

serialize - To ensure that one or more events occur in a specified sequence.

serially reusable resource (SRR) - A logical resource or object that can be accessed by only one task at a time.

session - (1) A routing mechanism for user interaction via the console; a complete environment that determines how an application runs and how users interact with the application. OS/2 can manage more than one session at a time, and more than one process can run in a session. Each session has its own set of environment variables that determine where OS/2 looks for dynamic-link libraries and other important files. (2) (D of C) In the OS/2 operating system, one instance of a started program or command prompt. Each session is separate from all other sessions that might be running on the computer. The operating system is responsible for coordinating the resources that each session uses, such as computer memory, allocation of processor time, and windows on the screen.

Settings Notebook - A control window that is used to display the settings for an object and to enable the user to change them.

shadow - An object that refers to another object. A shadow is not a copy of another object, but is another representation of the object.

shadow box - The area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shared data - Data that is used by two or more programs.

shared memory - In the OS/2 operating system, a segment that can be used by more than one program.

shear - In computer graphics, the forward or backward slant of a graphics symbol or string of such symbols relative to a line perpendicular to the baseline of the symbol.

shell - (1) A software interface between a user and the operating system of a computer. Shell programs interpret commands and user interactions on devices such as keyboards, pointing devices, and touch-sensitive screens, and communicate them to the operating system. (2) Software that allows a kernel program to run under different operating-system environments.

shutdown - The process of ending operation of a system or a subsystem, following a defined procedure.

sibling processes - Child processes that have the same parent process.

sibling windows - Child windows that have the same parent window.

simple list - A list of like values; for example, a list of user names. Contrast with *mixed list* .

single-byte character set (SBCS) - A character set in which each character is represented by a one-byte code. Contrast with *double-byte character set* .

slider box - In SAA Advanced Common User Access architecture: a part of the scroll bar that shows the position and size of the visible information in a window relative to the total amount of information available. Also known as *thumb mark* .

SOM - System Object Model.

source file - A file that contains source statements for items such as high-level language programs and data description specifications.

source statement - A statement written in a programming language.

specific dynamic-link module - A dynamic-link module created for the exclusive use of an application.

spin button - In SAA Advanced Common User Access architecture, a type of entry field that shows a scrollable ring of choices from which a user can select a choice. After the last choice is displayed, the first choice is displayed again. A user can also type a choice from the scrollable ring into the entry field without interacting with the spin button.

spline - A sequence of one or more Bézier curves.

spooler - A program that intercepts the data going to printer devices and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents output from different sources from being intermixed.

stack - A list constructed and maintained so that the next data element to be retrieved is the most recently stored. This method is characterized as last-in-first-out (LIFO).

standard window - A collection of window elements that form a panel. The standard window can include one or more of the following window elements: sizing borders, system menu icon, title bar, maximize/minimize/restore icons, action bar and pull-downs, scroll bars, and client area.

static control - The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

static storage - (1) A read/write storage unit in which data is retained in the absence of control signals. (A) Static storage may use dynamic addressing or sensing circuits. (2) Storage other than *dynamic storage* . (A)

style - See *window style* .

subclass - A class that inherits from another class. See also *Inheritance* .

subdirectory - In an IBM personal computer, a file referred to in a root directory that contains the names of other files stored on the diskette or fixed disk.

superclass - A class from which another class inherits. See also *inheritance* .

swapping - (1) A process that interchanges the contents of an area of real storage with the contents of an area in auxiliary storage. (I) (A)
(2) In a system with virtual storage, a paging technique that writes the active pages of a job to auxiliary storage and reads pages of another job from auxiliary storage into real storage. (3) The process of temporarily removing an active job from main storage, saving it on disk, and processing another job in the area of main storage formerly occupied by the first job.

switch - (1) In SAA usage, to move the cursor from one point of interest to another; for example, to move from one screen or window to another or from a place within a displayed image to another place on the same displayed image. (2) In a computer program, a conditional instruction and an indicator to be interrogated by that instruction. (3) A device or programming technique for making a selection, for example, a toggle, a conditional jump.

switch list - See *Task List* .

symbolic identifier - A text string that equates to an integer value in an include file, which is used to identify a programming object.

symbols - In Information Presentation Facility, a document element used to produce characters that cannot be entered from the keyboard.

synchronous - Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T)
See also *asynchronous* .

System Menu - In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

System Object Model (SOM) - A mechanism for language-neutral, object-oriented programming in the OS/2 environment.

system queue - The master queue for all pointer device or keyboard events.

system-defined messages - Messages that control the operations of applications and provides input an other information for applications to process.

Systems Application Architecture (SAA) - A set of IBM software interfaces, conventions, and protocols that provide a framework for designing and developing applications that are consistent across systems.

Glossary - T

table tags - In Information Presentation Facility, a document element that formats text in an arrangement of rows and columns.

tag - (1) One or more characters attached to a set of data that contain information about the set, including its identification. (I) (A) (2) In Generalized Markup Language markup, a name for a type of document or document element that is entered in the source document to identify it.

target object - An object to which the user is transferring information.

Task List - In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

terminate-and-stay-resident (TSR) - Pertaining to an application that modifies an operating system interrupt vector to point to its own location (known as hooking an interrupt).

text - Characters or symbols.

text cursor - A symbol displayed in an entry field that indicates where typed input will appear.

text window - Also known as the VIO window.

text-windowed application - The environment in which the operating system performs advanced-video input and output operations.

thread - A unit of execution within a process. It uses the resources of the process.

thumb mark - The portion of the scroll bar that describes the range and properties of the data that is currently visible in a window. Also known as a *slider box*.

thunk - Term used to describe the process of address conversion, stack and structure realignment, etc., necessary when passing control between 16-bit and 32-bit modules.

tilde - A mark used to denote the character that is to be used as a mnemonic when selecting text items within a menu.

time slice - (1) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing-unit time is allocated to another task, so a task cannot monopolize processing-unit time beyond a fixed limit. (2) In systems with time sharing, a segment of time allocated to a terminal job.

time-critical process - A process that must be performed within a specified time after an event has occurred.

timer - A facility provided under the Presentation Manager, whereby Presentation Manager will dispatch a message of class WM_TIMER to a particular window at specified intervals. This capability may be used by an application to perform a specific processing task at predetermined intervals, without the necessity for the application to explicitly keep track of the passage of time.

timer tick - See *clock tick*.

title bar - In SAA Advanced Common User Access architecture, the area at the top of each window that contains the window title and system menu icon. When appropriate, it also contains the minimize, maximize, and restore icons. Contrast with *panel title*.

TLB - Translation lookaside buffer.

transaction - An exchange between a workstation and another device that accomplishes a particular action or result.

transform - (1) The action of modifying a picture by scaling, shearing, reflecting, rotating, or translating. (2) The object that performs or defines such a modification; also referred to as a *transformation*.

Translation lookaside buffer (TLB) - A hardware-based address caching mechanism for paging information.

Tree - In the Presentation Manager, the window in the *File Manager* that shows the organization of drives and directories.

truncate - (1) To terminate a computational process in accordance with some rule (A) (2) To remove the beginning or ending elements of a string. (3) To drop data that cannot be printed or displayed in the line width specified or available. (4) To shorten a field or statement to a specified length.

TSR - Terminate-and-stay-resident.

unnamed pipe - A circular buffer, created in memory, used by related processes to communicate with one another. Contrast with *named pipe*.

unordered list - In Information Presentation Facility, a vertical arrangement of items in a list, with each item in the list preceded by a special character or bullet.

update region - A system-provided area of dynamic storage containing one or more (not necessarily contiguous) rectangular areas of a window that are visually invalid or incorrect, and therefore are in need of repainting.

user interface - Hardware, software, or both that allows a user to interact with and perform operations on a system, program, or device.

User Shell - A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

utility program - (1) A computer program in general support of computer processes; for example, a diagnostic program, a trace program, a sort program. (T) (2) A program designed to perform an everyday task such as copying data from one storage device to another. (A)

Glossary - U

There are no glossary terms for this starting letter.

Glossary - V

value set control - A visual component that enables a user to select one choice from a group of mutually exclusive choices.

vector font - A set of symbols, each of which is created as a series of lines and curves. Synonymous with *outline font* . Contrast with *image font* .

VGA - Video graphics array.

view - A way of looking at an object's information.

viewing pipeline - The series of transformations applied to a graphic object to map the object to the device on which it is to be presented.

viewing window - A clipping boundary that defines the visible part of model space.

VIO - Video Input/Output.

virtual memory (VM) - Synonymous with *virtual storage* .

virtual storage - (1) The storage space that may be regarded as addressable main storage by the user of a computer system in which virtual addresses are mapped into real addresses. The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of auxiliary storage available, not by the actual number of main storage locations. (I) (A) (2) Addressable space that is apparent to the user as the processor storage space, from which the instructions and the data are mapped into the processor storage locations. (3) Synonymous with *virtual memory* .

visible region - A window's presentation space, clipped to the boundary of the window and the boundaries of any overlying window.

volume - (1) A file-system driver that uses a block device driver for input and output operations to a local or remote device. (I) (2) A portion of data, together with its data carrier, that can be handled conveniently as a unit.

Glossary - W

wildcard character - Synonymous with *global file-name character* .

window - (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area of the screen with visible boundaries within which information is displayed. A window can be smaller than or the same size as the screen. Windows can appear to overlap on the screen.

window class - The grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates - A set of coordinates by which a window position or size is defined; measured in device units, or *pels* .

window handle - Unique identifier of a window, generated by Presentation Manager when the window is created, and used by applications to direct messages to the window.

window procedure - Code that is activated in response to a message. The procedure controls the appearance and behavior of its associated windows.

window rectangle - The means by which the size and position of a window is described in relation to the desktop window.

window resource - A read-only data segment stored in the .EXE file of an application or the .DLL file of a dynamic link library.

window style - The set of properties that influence how events related to a particular window will be processed.

window title - In SAA Advanced Common User Access architecture, the area in the title bar that contains the name of the application and the OS/2 operating system file name, if applicable.

Workplace Shell - The OS/2 object-oriented, graphical user interface.

workstation - (1) A display screen together with attachments such as a keyboard, a local copy device, or a tablet. (2) (D of C) One or more programmable or nonprogrammable devices that allow a user to do work.

world coordinates - A device-independent Cartesian coordinate system used by the application program for specifying graphical input and output. (I) (A)

world-coordinate space - Coordinate space in which graphics are defined before transformations are applied.

WYSIWYG - What-You-See-Is-What-You-Get. A capability of a text editor to continually display pages exactly as they will be printed.

Glossary - X

There are no glossary terms for this starting letter.

Glossary - Y

There are no glossary terms for this starting letter.

Glossary - Z

z-order - The order in which sibling windows are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zooming - The progressive scaling of an entire display image in order to give the visual impression of movement of all or part of a display group toward or away from an observer. (I) (A)

8.3 file-name format - A file-naming convention in which file names are limited to eight characters before and three characters after a single dot. Usually pronounced "eight-dot-three." See also *non-8.3 file-name format* .
